Spatial Network Big Data: Challenges, Approaches, and Opportunities

A DISSERTATION
SUBMITTED TO THE FACULTY OF THE GRADUATE SCHOOL
OF THE UNIVERSITY OF MINNESOTA
BY

KwangSoo Yang

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
Doctor of Philosophy

Advisor: Professor Shashi Shekhar

May, 2015

Acknowledgements

I would like to express my sincere appreciation and thanks to my advisor Professor Shashi Shekhar. His enthusiasm, inspiration, and great efforts helped me to grow as a research scientist. His advice on both research as well as on my career have been priceless. I would also like to express my gratitude to my committee members, professor Ravi Janardan, professor George Karypis, professor Jarvis Haupt for their helpful suggestions and for serving on my thesis committee. I also extend my appreciation to the University of Minnesota Spatial Computing Research Group for their brilliant comments and suggestions. Lastly, I thank my mother, sister, brother, and the Lundberg family for everything they have done for me.

Abstract

Spatial Network Big Data (SNBD) refers to spatial network datasets whose size, variety, or update rate exceeds the capacity of commonly-used spatial network computing and spatial network database technologies to learn, manage, and process with reasonable effort. SNBD has the potential to transform society via next-generation routing services, emergency and disaster response, and discovery of potentially useful patterns embedded in these datasets. However, the enormous complexity of SNBD raises many computer science challenges. My research aims to address these challenges via applying novel SNDB database systems to effectively harness the power of SNBD. This thesis studied three challenging SNDB database problems.

To address the challenge of query processing for resource and shelter allocation in the wake of man-made and natural disasters, we investigated the problem of Capacity-Constrained Network-Voronoi Diagram (CCNVD). Given a graph and a set of service center nodes, CCNVD partitions the graph into a set of contiguous service areas that meet service center capacities and minimize the sum of the shortest distances from graph-nodes to allotted service centers. The CCNVD problem is important for critical societal applications such as assigning evacuees to shelters and assigning patients to hospitals. This problem is NP-hard; it is computationally challenging because of the large size of the transportation network and the constraint that service areas must be contiguous in the graph to simplify communication of allotments. Previous work has focused on honoring either service area contiguity (e.g., Network Voronoi Diagrams) or service center capacity constraints (e.g., min-cost flow), but not both. We proposed novel Pressure Equalizer (PE) approaches for CCNVD to meet the capacity constraints of service centers while maintaining the contiguity of service areas. Experiments using road maps from five different regions demonstrate that the proposed approaches significantly reduce computational cost.

To address the challenge of query processing for traffic congestion and choke-points during or after disasters, we explored the problem of Evacuation Route Planning (ERP). Given a transportation network, a population, and a set of destinations, the goal of evacuation route planning is to produce routes that minimize the evacuation time for the

population. Evacuation planning is essential for ensuring public safety in the wake of man-made or natural disasters (e.g., terrorist acts, hurricanes, and nuclear accidents). The problem is challenging because of the large size of network data, the large number of evacuees, and the need to account for capacity constraints in the road network. Promising methods that incorporate capacity constraints into route planning have been developed but new insights are needed to reduce the high computational costs incurred by these methods with large-scale networks. In this work, we propose a novel scalable approach that explicitly exploits the spatial structure of road networks to minimize the computational time. Our new approach accelerates the routing algorithm by partitioning the network using dartboard network-cuts and groups node-independent shortest routes to reduce the number of search iterations. Experimental results using a Minneapolis, MN road network demonstrate that the proposed approach significantly reduces the computational cost for evacuation route computation.

To address the challenge of storing Spatio-Temporal Networks (STN), we explored the problem of Storing Spatio-Temporal Networks (SSTN). Given a spatio-temporal network (STN) and a set of STN operations, the goal of SSTN is to find a storage scheme that minimizes the I/O costs of the operations. The SSTN problem is important for many societal applications such as surface and air transportation management systems. The problem is NP hard, and is challenging due to an inherently large data volume and novel semantics (e.g., Lagrangian reference frame). Related works rely on orthogonal partitioning approaches (e.g., snapshot and longitudinal) and incur excessive I/O costs when performing common STN queries. In this work, we proposed novel non-orthogonal partitioning approaches in which we optimize the STN operation for a given node on an STN. Experimental results using real-world road and flight traffic datasets demonstrate that the proposed approaches outperform prior work for STN query computation workloads.

The work in this thesis is the first step towards understanding the immense challenges and novel applications of SNBD database systems. In this thesis, we have formally modeled two query processing strategies (i.e., CCNVD and ERP) and begun to explore scalable algorithms to minimize the computational cost for query processing. We have also investigated a method of storing SNBD and studied how to develop I/O efficient storage and access methods. Possible directions for future work include SNBD Logical

Data Model, SNBD Query Language, SNBD Query Processing Strategy, and SNBD Storage Model.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Increasingly, Spatial Network Big Data (SNBD) is of a size, variety, or update rate that exceeds the capacity of commonly-used spatial computing technologies to learn, manage, and process with reasonable effort [1]. Examples of SNBD include temporally detailed road maps that provide speeds every minute for every road-segment, GPS trace data from cell-phones, and engine measurements of fuel consumption, greenhouse gas emissions, etc.

1.1 Societal Importance

Spatial Network Big Data has the potential to transform our society. For example, a 2011 McKinsey Global Institute report estimates savings of about $600 billion annually by 2020 in terms of fuel and time saved by helping vehicles avoid congestion and reduce idling at red lights or left turns [2]. WAZE and Uber are already helping us route around congestion and quickly find taxis [3, 4]. Scientist are investigating SNBDs for hypothesis generation to address complex urban questions, where progress before was hampered by data paucity.

Below I describe these application domains in SNBD: resource and shelter allotment, evacuation route planning, and surface and air transportation management systems.

1.1.1 Resource and Shelter Allotment

The Capacity Constrained Network Voronoi Diagram (CCNVD) problem is important for critical applications such as assigning people to relief-supply distribution centers in the aftermath of a disaster, assigning evacuees to shelter facilities, and assigning jurisdictions to emergency responders such as hospitals and police stations. For example, after hurricane Sandy, New Jersey residents had to wait for hours to fill up car tanks and containers for home generators [5]. Such fuel shortages serve as a reminder of the importance of resource allotment amid natural or man-made disasters such as floods, hurricanes, tsunamis, fires, terrorist acts, and industrial accidents. Therefore, an important aspect of disaster response is to define service areas for facilities, e.g., gas stations, shelters, hospitals, etc. One challenge is to model the capacities of service centers to reduce overloading. People naturally go to their nearest service centers, even if service centers farther away have no wait. Emergency managers need tools to assist them with service area delineation, which we conceptualize in our work as the Capacity Constrained Network Voronoi Diagram (CCNVD) problem. CCNVD ensures the safety of the people (or cars) by providing conflict-free and short routes to access exits (or shelters) in an overcrowded area. For example, CCNVD may also be used for shelter allocation where contiguous service areas reduce movement conflicts (which raise risk of congestion, stampede, etc.) across people heading to different shelters. It also ensures that communications clearly explain the emergency information of official instructions by providing contiguous service areas.

1.1.2 Evacuation Route Planning

Hurricane Rita and the recent Tohuku tsunami that hit Japan are reminders that evacuation planning is an essential component of civic emergency preparedness. Ensuring the safety of all residents of a structure, city, or region during a disaster requires evacuation planning tools to produce the safest and most efficient route schedules for large scale road networks and populations within limited time constraints. Consider a hurricane evacuation planning problem. Low lying riverside and coastal regions, are especially at risk for a major storm or flooding. The speed and direction of a hurricane can change rapidly, so the threat to particular areas of the coast may come up suddenly. Massive

emergency evacuation from these areas brings more challenges for civic authorities due to the large and unpredictable shape of evacuation zones (EZs) along coastal areas. In 2005, the approach of hurricane Rita provoked one of the largest evacuations in U.S. history, resulting in three million evacuees. During the evacuation, the enormous number of people fleeing from the Houston area coupled with a number of shortcomings in exit routes for residents caused massive traffic jams. In 1992, Hurricane Andrew, the third most powerful storm to hit the Florida coast caused massive delays and major congestion. Previously, disasters like Rita and Andrew demonstrated the inadequacy of hand drawn plans for evacuating populations after a disaster. They also demonstrated the need to account for the capacity constraints of road networks. Computational methods of evacuation planning promise more efficient route schedules in the face of massive storms. These methods must be scalable and able to produce results easily in a short time frame. Furthermore, they must be able to handle dynamic environments.

1.1.3 Surface and air transportation management systems

Spatial Network Big Data (SNBD) requires next-generation storage and access methods for Spatio-Temporal Networks (STN) that minimizes disk I/Os and performs STN operations or spatial computing for big STN data sets (e.g., traffic, GPS). For example, the Federal Highway Administration [6] is recording traffic data on major roads and highways using sensors, such as loop detectors, across the United States. Depending on the type of sensor, traffic levels are recorded as often as every minute or less. The Mobility Monitoring Program (MMP), started in 2000 by the Texas Transportation Institute, evaluated the use of sensors for traffic information around the United States. By 2003, the MMP was receiving traffic sensor data from over 30 cities and $3,000$ miles of highway, with sensor readings occurring roughly every 30 seconds. These data are recorded 24 hours a day, 365 days a year, resulting in millions of time steps per year for each sensor. In 2004, MMP published a report citing the need for better processing and storage of historical traffic data in order to benefit traffic management [6]. As an another example, airlines connect thousands of destinations across the world through various routes between airports. Maintaining accurate records of route performance is essential to evaluating and ensuring timely airline service, along with analyzing the potential causes and effects of delays. In order to measure route characteristics, such

as average delay, each flight along the route is recorded with parameters such as flight time, departure time, landing time, etc. This flight information creates an STN that allows historical queries to be answered, such as how often a flight is 'on time', 'late', 'very late', or 'excessively late'. Other more complex queries, such as how a delay on a particular route affects connecting flights, can also be analyzed with this data.

1.2 Proposed Approaches

This thesis investigates novel approaches for SNBD database management systems. For example, consider three levels of SNBD database management systems shown in Table 1.1. The conceptual level provides a SNBD conceptual data model, which helps us to define essential requirements and processes in database design. SNBD conceptual data model is a high-level description of the data requirements of the users (e.g., entity types, relationships, and constraints). Examples of SNBD conceptual data models include Entity-Relationship Diagrams (ERD) and pictograms. However, these models have a limitation to fully express the spatial and spatio-temporal network structures (e.g., spatio-temporal topological relationship, spatio-temporal hotspots, etc.).

Table 1.1: Three levels of SNBD database management systems

Level	Sub-component	Thesis Chapter
Conceptual	SNBD Conceptual Data Model	
Logical	SNBD Logical Data Model	
	SNBD Query Language	
Physical	SNBD Query Processing Strategy	Chapter 2: Capacity Constrained Network Voronoi Diagram (CCNVD) Chapter 3: Evacuation Route Planning (ERP)
	SNBD Storage Model	Chapter 4: Lagrangian Approaches to Storage of Spatio-temporal Network Big Data (STNBD)

The logical level transforms the high-level data model into the implementation data model. SNBD logical data model provides a detailed data structure of a domain of information (e.g., tables, views, columns, primary keys, foreign keys, etc.). Examples of SNBD logical data models include Time-Expanded Graph (TEG) and Time-Aggregated Graph (TAG) [7, 8]. However, both TEG and TAG face a challenge to represent big spatio-temporal networks because of the large size of network data and the large number

of time-points. In addition, the logical level requires query languages that operate over spatial networks. Examples of query language include Structured Query Language (SQL) for relational databases and Object Query Language (OQL) and SQL3 for object databases [9]. However, these existing query languages do not fully support spatial network queries (e.g., shortest path computation, network flow computation, Network Voronoi Diagrams, etc.).

The physical level transforms the implementation data model into an equivalent actual physical data model (e.g., database files, indexes, access paths, query processing, etc.). This physical level consists of two main sub-components: SNBD query processing strategies and SNBD storage models. The main goal of SNBD query processing strategy is to minimize the execution time of spatial network data processing to answer a query. Examples of these strategies include network-traversal algorithms (e.g., breadth-first search, depth-first search, shortest path computation, etc.), network flow algorithms (e.g., maximum flow computation, minimum cost flow computation, etc.), and network partitioning algorithms (e.g., min-cut graph partitioning, Network-Voronoi Diagrams, etc.). SNBD storage model is a storage scheme which describes data-structures, storage and access methods, and indexes. Storage model involves deep use of particular database management technology, such as data clustering and I/O efficient indexing. Examples of SNBD storage model include physical data models (e.g., node, edge, connection, etc.), index data-structures, network access methods (e.g., $getOneSuccessor()$, $getAllSuccessors()$, etc.).

1.3 Challenges

SNBD raises many computer science challenges. First, current spatial network database storage methods are challenged by their temporal graph-based query semantics as well as the growing volume of temporally-detailed road-maps. In addition, query processing methods are challenged by emerging use cases of spatial resource assignment, such as service centers and routes.

This thesis aims to address these challenges via investigating novel approaches based on the idea of scalable graph-based query processing strategies and I/O efficient storage and access methods.

1.3.1 Computational Challenges

Spatial Network Big Data (SNBD) requires new computational strategies for graph-based query processing. First I explore novel computational techniques for creating a Capacity Constrained Network Voronoi Diagram (CCNVD). CCNVD allots routes (e.g., evacuation routes) or limited resources (e.g., gas, water, or shelters) equally efficiently and more safely to evacuees (or consumers). One of the biggest challenges in this problem is to minimize the computational cost to analyze SNBD and quickly respond to emergency situations. This problem is NP-hard; it is computationally challenging because of the large size of the transportation network and the constraint that service areas must be contiguous. Second, I investigate an Evacuation Route Planning (ERP) problem that produces routes and minimizes the evacuation time for a population after a man-made or natural disaster (e.g., terrorist act, hurricane, and nuclear accident). The problem is challenging because of the large size of SNBD, the large number of evacuees, and the need to account for capacity constraints in the road network. Promising methods that incorporate capacity constraints into route planning have been developed but new insights are needed to reduce the high computational costs incurred by these methods with large-scale SNBD [10, 11, 12].

Emerging large-sized Spatio-temporal Network datasets require novel data storage and access methods. Given a spatio-temporal network (STN) and a set of STN operations, the goal of the Storing Spatio-Temporal Networks (SSTN) problem is to produce an efficient method of storing STN data that minimizes disk I/O costs for given STN operations. The problem is NP-hard, and is challenging due to an inherently large data volume and novel semantics (e.g., Lagrangian reference frame). The Lagrangian frame of reference requires new data types, query operations, and storage management systems to coordinates STN datasets with STN connectivity and efficiently store and query STN datasets. STN datasets are usually massive in size and are accessed based on spatio-temporal movement patterns, making I/O efficient storage and access methods a significant challenge.

1.4 Thesis Contributions

This thesis makes three main contributions outlined in the previous section.

The first is a Pressure Equalizer (PE) approach that creates a Capacity Constrained Network Voronoi Diagram (CCNVD) [13]. PE follows three main steps: (1) initial assignment of graph-nodes to their nearest service centers (i.e., creation of an Network Voronoi Diagram); (2) construction of a new data structure called a PE-graph with PE-nodes representing service centers and PE-edges between PE-nodes whose service areas are adjacent; and (3) re-allotment of graph-nodes from overloaded (excess) service centers to underloaded (deficit) service centers. The key idea underlying the PE approach is to consider only boundary nodes for the re-allotment. In addition, we propose a novel Block Tree Contiguity Checking algorithm (BTCC) to reduce the computation cost of the PE approach. The BTCC achieves a significant computational performance improvement because it creates a Block Tree and checks the Service Area contiguity with Look-up tables in constant time. Experimental results with real road network datasets showed that our proposed PE approaches significantly reduced the computation cost to create a CCNVD.

The second contribution is a dartboard network-cut for evacuation route planning (DBNC-ERP) algorithm that produces routes that minimize the evacuation time for a population [11]. DBNC-ERP uses an underlying dartboard network structure driven by DBN-cuts. DBN-cuts group multiple node-independent shortest routes and reduce iterations of an evacuation routing algorithm. We experimentally evaluated the proposed algorithm and validate the cost model using real road network datasets.

Finally, a non-orthogonal partitioning approach is presented to optimize STN operations [8]. I focus on a special case of the SSTN problem, namely SSTN-G∀S, that optimizes the $LGetAllSuccessors()$ operation for retrieving all successors for a given node on an STN. The SSTN-G∀S problem is NP-hard and is computationally challenging because of the fixed data page size, the large size of STN datasets, and the constraint that data page access for the STN operation must be minimized. We developed a novel storage and access method, namely LCP-G∀S, using the concept of a Lagrangian Family Set (LFS) that created a solution to the SSTN-G∀S problem. We experimentally evaluated the proposed approach using real-world and synthetic STN datasets.

1.5 Scope

My research focuses on issues relating to the physical level of SNBD database systems. First, we investigated an emergency logistics planning problem, namely Capacity-constrained Network Voronoi Diagram (CCNVD), to designate service areas for shelter or centers distributing relief supplies, e.g., gas, food, water. Second, we studied the computational question in the context of Evacuation Route Planning (ERP). Given a transportation network, a population, and a set of destinations, the goal of ERP is to produce routes that minimize the evacuation time for the population. ERP is essential for ensuring public safety in the wake of man-made or natural disasters (e.g., terrorist acts, hurricanes, and nuclear accidents). Finally, we explored efficient storage methods for new generation temporally-detailed roadmaps showing time-variation of speed over different time points in a typical week. Given a spatio-temporal network (STN) and a set of STN operations, the problem of Storing Spatio-Temporal Networks (SSTN) finds a storage scheme that minimizes the I/O costs of the STN operations.

1.6 Thesis Organization

The rest of the thesis is organized as follows: Chapter 2 introduces the Capacity Constrained Network Voronoi Diagram (CCNVD) problem and describes our proposed approaches to create a CCNVD. Chapter 3 introduces the Evacuation Route Planning (ERP) problem and our proposed approaches to produce evacuation routes for large scale network datasets. Chapter 4 introduces the Storing Spatio-Temporal Networks (SSTN) problem and describes our proposed approaches to store and access massive STN datasets. Finally, Chapter 5 summarizes our findings and identifies related areas that remain open for future research.

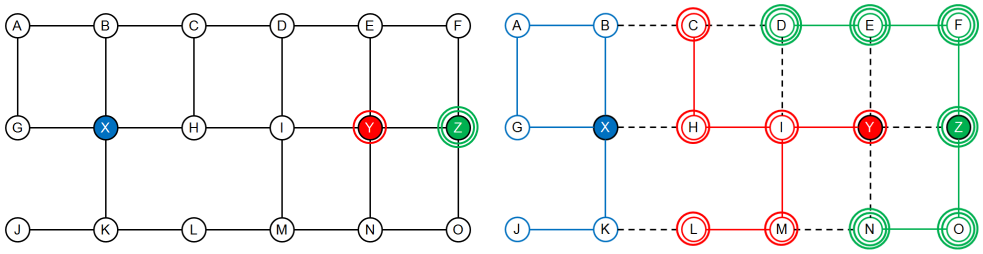
Chapter 2

Capacity Constrained Network Voronoi Diagram

2.1 Introduction

Given a graph and a set of service center nodes (e.g., gas stations) with capacity constraints (e.g., amount of gasoline, size of parking lot, etc.), a Capacity Constrained Network-Voronoi Diagram (CCNVD) partitions the graph into a set of contiguous service areas (SAs) that honor service center capacities and minimize the sum of the shortest distances from graph-nodes to allotted service centers. Figure 2.1(a) shows an example input of CCNVD consisting of a graph with 15 graph-nodes ($A, B, \ldots, O$) and three service center nodes (X, Y, and Z) with capacities of 5 each. Figure 2.1(b) shows an example output of CCNVD where the graph is partitioned such that 5 graph-nodes are allotted to each service center, as shown by the distinct colors. In essence, the research problem is to construct a CCNVD that assigns users to service centers while meeting contiguity and capacity constraints and minimizing the total shortest distance from users to assigned facilities.

The CCNVD problem is NP-hard (a proof is provided in Section 2.1.4). Intuitively, the problem is computationally challenging because of the large size of the transportation network and the contiguity constraint.



(a) Input (b) Output

Figure 2.1: Example of the Input and Output of CCNVD (Colors show service center allotment)

2.1.1 Application Domain

The CCNVD problem is important for critical applications such as assigning people to relief-supply distribution centers in the aftermath of a disaster, assigning evacuees to shelter facilities, and assigning jurisdictions to emergency responders such as hospitals and police stations. CCNVD ensures the safety of the people (or cars) by providing conflict-free and short routes to access exits (or shelters) in an overcrowded area. It also ensures that communications clearly explain the emergency information of official instructions by providing contiguous service areas. Examples of such situations are provided in Table 2.1. In all examples, the CCNVD objective, constraints, and computation time are important.

CCNVD objective is to minimize the sum of the shortest distances from drivers to their assigned service centers. Distance affects the travel time from user to the assigned facility, impacting the speed of evacuation or the speed of emergency response. The capacity constraint helps facilities run efficiently. If a facility were burdened with more users than it could accommodate, it would have wait times and may not have enough resources for evacuees, or first responders for disasters. The contiguity constraint has two benefits: (1) Contiguous area assignments are easier to communicate (assigning several disconnected areas to one facility is harder to communicate than assigning one connected region to the facility) as well as to follow (criss-crossing paths may divert people to the wrong facility). (2) Contiguity constraint reduces movement-conflict across people heading to different service centers.

The importance of travel distance, capacity, and contiguity were clearly evident in

Table 2.1: Applications of CCNVD

Application	Benefit of Contiguous Service Areas
Relief Distribution	Reduce movement-conflicts across people heading to different relief-supply distribution centers.
Mass Evacuation	Reduce criss-crossing across people heading to different shelters/exits.
Mass Vaccination	Simplify communication of service areas of different vaccination centers.
School	Reduce human-human collisions across students heading to different exits.
Airport	Speed up evacuation of passengers heading to different security check points.
Post-game traffic management	Improve traffic flow by separating cars in parking ramp heading to different freeway-entrances.

the aftermath of Hurricane Sandy [5]. After the devastating storm, people in need of fuel naturally headed to their nearest gas stations. However, these stations often had excessive wait times due to gas shortages (Figure 2.2). Meanwhile, further inland, there were stations that had available gas. CCNVD assignment could have redirected fuel seekers to gas stations with available gas. The key observation is that people naturally go to the nearest gas station when they would be better served going to the nearest station with available gas.

The fact that CCNVDs can be computed and updated quickly, also enhances their value under disaster conditions. For example, in emergency or disaster situations, planners must often respond to unforeseen changes (e.g., a fallen tree or water blocking a road segment, which could be represented by the deletion of an edge in the road network). An algorithm that runs quickly could be rerun after each unforeseen change and can thus better adapt to specific circumstances than a slower algorithm.

Indeed, CCNVDs can be used for many types of resource allocation including allocation of evacuation shelters, hospital assignment, etc. Tools for efficient resource allocation in disaster situations enhance transportation resilience, that is, the ability of

(a) Vehicles wait in line for fuel at gas station (Courtesy: www.bloomberg.com)

(b) Waiting to fill containers for home generators (Courtesy: Andrew Burton/Getty)

Figure 2.2: Long lines at gas station after Hurricane Sandy in the New York-New Jersey area

transportation networks to prepare for respond to, and recover from significant, potentially multi-hazard threats with minimum impact on public safety, public health, the economy, and environment. There is growing concern about transportation resilience due to the increased frequency and greater magnitude of extreme events such as hurricanes. A 2013 Presidential climate change order aims to prepare communities for severe weather [14], and recent post-Hurricane Sandy action plans for New York City highlight the importance of improved evacuation planning, including updated evacuation zones and better communication [15]. Transportation resiliency is currently in the national spotlight and will only grow in importance as the frequency and magnitude of extreme events increase.

2.1.2 Problem Definition

In our formulation of the CCNVD problem, a transportation network is represented and analyzed as an undirected graph composed of nodes and edges. Each node represents a spatial location in geographic space (e.g., road intersections), which can be used as a proxy for locations of citizens or residences. Each edge between two nodes represents a road segment and has a travel distance. Each service center has a given capacity (the number of people it can efficiently serve). The $CCNVD(N, E, S, C, D)$ problem is defined as follows:

Input: A transportation network G with

- a set of graph-nodes N and a set of edges E,
- a set of fixed service center locations $S \subset N$,
- a set of positive integer capacities for service centers $C : S \rightarrow Z^+$, and
- a set of nonnegative real distances of edges $D : E \rightarrow R_0^+$

Output: A Capacity Constrained Network Voronoi Diagram (CCNVD)

Objective:

- Min-sum: Minimize the sum of the shortest distances from graph-nodes to their allotted service centers.

Constraints:

- Capacity Constraint: Each service area (SA) contains exactly one service center and the number of graph-nodes in the SA does not exceed the capacity of the SA.
- Contiguity Constraint: Each service area (SA) should be a connected sub-graph of G.

Relationship with Capacitated Facility Location Problem

The CCNVD problem is distinct from the well-known Capacitated Facility Location Problem (CFLP) [16, 17]. First, CCNVD is an assignment problem while CFLP is a location problem. In other words, CCNVD assigns users to existing facilities whose locations are fixed, while CFLP determines locations for new facilities where both the number and locations of facilities can be changed. Second, CCNVD has a contiguity constraint, while CFLP does not (primarily because CFLP does not assign users).

Other related problems including NVD and min-cost flow are discussed in Section 2.5.3.

Variations of CCNVD

The CCNVD problem focuses on minimizing the sum of the shortest distances. The shortest distance can be classified into two types: network shortest distance and service area (SA) shortest distance. The following definitions show the difference between them.

Definition 1. ***Network Shortest Distance***

Given a spatial network $G(N, E, D)$ and two nodes n and s, the network shortest distance is the shortest path distance from n to s in $G(N, E, D)$.

Definition 2. ***Service Area (SA) Shortest Distance***

Given a connected sub-graph $SA(N, E, D) \subset G(N, E, D)$ and two nodes n and s, the SA shortest distance is the shortest path distance from n to s in the SA.

2.1.3 Approximation Hardness

The NPO-completeness of CCNVD follows from a well-known result about the NP-hardness of the connected k-partition problem [18, 19, 20], which partitions a graph into k connected sub-graphs where $k \geq 3$.

Definition 3. ***Connected k-partition problem*** *[18, 19]*
Given a graph $G = (N, E)$, service centers $s_1, s_2, \ldots, s_k \in N$, and positive integer capacities for service centers $c_1, c_2, \ldots, c_k$, where $c_1 + c_2 + \ldots + c_k = |N|$, the connected-$k$-partition ($k{-}CP(N, E, S, C)$) problem separates the service centers $s_1, s_2, \ldots, s_k$, and partition p_i containing s_i is a connected sub-graph consisting of c_i nodes for $i = 1, 2, \ldots, k$ (e.g., equal sized connected sub-graphs). It has been proved that a connected-k-partition of N is a NP-hard problem [18].

Theorem 1. *No polynomial-time approximation algorithm exists for CCNVD if $P \neq NP$.*

Proof. Assume P $\neq$ NP. Let $n = |N|$ and $k = |S|$. We construct a mapping from an instance $k{-}CP(N, E, S, C)$ of the connected k-partition problem to an instance $CCNVD(N, E, S, C, D)$ of the CCNVD problem, such that the question of whether $k{-}CP$ has a solution can be determined from any $CCNVD$ whose min-sum is n. Given a $k{-}CP(N, E, S, C)$, we add additional edges to $k{-}CP$ and construct a complete graph $CG(N, E_{cg}, S, C)$. Then the instance of $CCNVD(N, E_{cg}, S, C, D)$ can be constructed by assigning $D(e)$ on every $e(v_i, v_j) \in E_{cg}$ as follows:

$$D(e) = \begin{cases} 1 & \text{if } e \in E \text{ and } (v_i \in S \text{ or } v_j \in S) \\ 1 + n \cdot g & \text{if } e \notin E \\ 0 & \text{otherwise} \end{cases}$$

If the min-sum of $CCNVD$ is n, then the solution of $k{-}CP(N, E, S, C)$ exists because all nodes in $SA(s \in S)$ form a connected component. If the min-sum of $CCNVD$ is at least $n \cdot (1 + g)$, then no solution of $k{-}CP$ exists because at least one node is disconnected from $SA(s \in S)$. If there exists a polynomial time g approximation algorithm for $CCNVD$, then it can solve the NP-hard problem $k{-}CP$ in polynomial time, implying P=NP. This contradicts the original assumption that P $\neq$ NP, so no polynomial-time approximation exists for CCNVD. $\square$

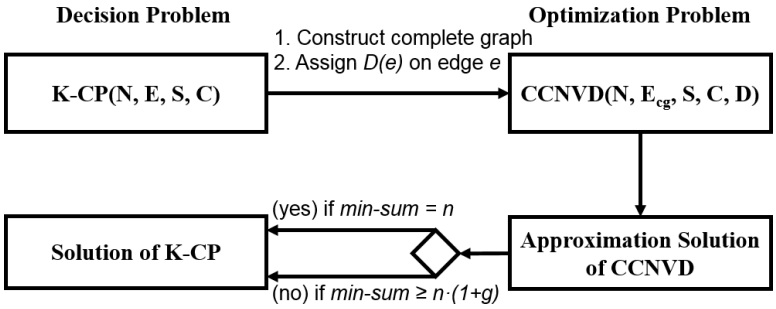


Figure 2.3: Approximation hardness of CCNVD

2.1.4 Problem Hardness

Theorem 2. *The CCNVD problem is NP-hard.*

Proof. The CCNVD problem belongs to NP since, given an instance of a CCNVD and a maximum bound T, we can take a set of connected sub-graphs such that the sum of the shortest distances from nodes (N) to their allotted service centers (S) is lower than T as a valid certification. Let $A = (N, E, S, C)$ be an instance of a connected-k-partition problem, where N is a set of nodes, E is a set of edges, S is a set of service centers, and C is a set of capacities for service centers. Let $B = (N, E, S, C, D, T)$ be an instance of the CCNVD problem, where D is a set of distances of E, and T is a maximum bound of the sum of the shortest distances from nodes (N) to their allotted service centers (S). Then it is easy to show that a connected k-partition is a special case of CCNVD, where k is the number of service centers S, $d \in D$ has a distance value of zero, and T is unbounded. Since A is constructed from B in polynomial time, the proof is complete. □

2.1.5 Relationship with Network Voronoi Diagram

Network Voronoi Diagram (NVD) is a special case of the CCNVD problem where the service center capacity is unlimited (see Lemma 1 and Lemma 2). The NVD problem can be formalized as follows: given the input of $G(E, N, D)$, the objective is to assign every graph-node to its nearest service center.

Lemma 1. *NVD can create contiguous service areas.*

Proof. Let d be a shortest path distance, and let $NVD(S)$ be the NVD of a finite set $S \subset N$ with respect to d. Then each service area $SA(s \in S)$ can be represented by a shortest path tree. Assume that $SA(s)$ is not a connected graph. Then it contradicts that a tree is a connected graph. □

Lemma 2. *NVD offers the optimal min-sum of the shortest distances when there is no capacity constraint.*

Proof. According to Lemma 1, every service area $SA(s \in S)$ can be represented by a shortest path tree. If we choose any graph-node $n \in SA(s1)$ and re-assign node n to $SA(s2)$, then the sum of the shortest distances is non-decreasing. Therefore, an NVD offers the optimal min-sum of the shortest distances. □

Figure 2.4 show an example of CCNVD. Every edge is associated with a distance, as indicated by the number displayed above it. In this example, the sum of the network shortest distances is 30 and the sum of the SA shortest distances is 32.

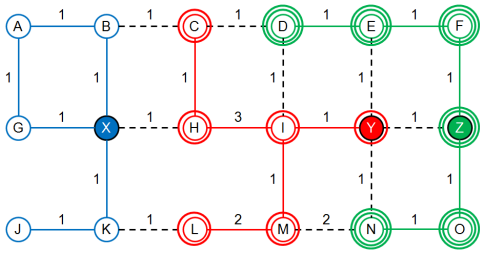


Figure 2.4: Example of a CCNVD where the network shortest distance is less than the SA shortest distance (SA min-sum=32, min-sum=30)

Using the network shortest distance increases the likelihood of travelers having to pass other service areas to reach their assigned service center. This may increase criss-crossing of the shortest paths, leading to congestion. However, the network shortest distance is a good approximation of the SA shortest distance and the optimization with this is more practical in terms of computation efficiency (see Section 2.2.1).

2.1.6 Our contributions

To the best of our knowledge, there is no published algorithm to solve the CCNVD problem. Our previous work proposed the naive Pressure Equalizer (PE) algorithm, namely

PE-SSTD, to address CCNVD, as reviewed in Section 2.2 [13]. PE-SSTD follows three main steps: (1) initial assignment of graph-nodes to their nearest service centers (i.e., creation of an NVD); (2) construction of a new data structure called a PE-graph with PE-nodes representing service centers and PE-edges between PE-nodes whose service areas are adjacent; and (3) re-allotment of graph-nodes from overloaded (excess) service centers to underloaded (deficit) service centers.

Our previous contributions were as follows:

- We proved that the CCNVD problem is NP-hard.
- We proposed PE-SSTD that creates CCNVD.
- Our experimental results and a case study demonstrated that PE-SSTD has comparable solution quality (in terms of min-sum) to min-cost flow, but maintains contiguity and has a significantly smaller computation cost compared to min-cost flow.

The PE-SSTD algorithm used a linear-time graph traversal algorithm (i.e., Breadth First Search (BFS)) to test whether service areas are contiguous. However, we find that this Service Area Contiguity Checking (SACC) is the main bottleneck of the PE-SSTD algorithm.

In this paper, we propose a novel Block Tree Contiguity Checking algorithm (BTCC) to reduce the computation cost of the PE-SSTD algorithm. In addition, we propose a more scalable solution that uses a Graph Minor by trading quality for efficiency. Finally, we experimentally and theoretically evaluate all PE algorithms.

In summary, our new contributions are as follows:

- We propose a novel Block Tree Contiguity Checking (BTCC) algorithm to reduce the computational cost of the Service Area Contiguity Checking (SACC).
- We propose a more scalability solution that uses a Graph Minor by trading quality for efficiency.
- We theoretically evaluate all proposed algorithms through cost models and proofs of algorithm properties (e.g., termination, correctness).
- We experimentally evaluate all proposed algorithms, as well as min-cost flow, using five different real-world road maps.

2.1.7 Scope and outline

In constructing our novel algorithm for CCNVD, we assume undirected edges, unit demand at each non-service-center node (graph-node), and no edge-capacity constraints. The validity of these assumptions may be addressed in future work. Additionally, in this work, the locations of service centers are known a priori and the goal is to create a CCNVD based on these locations. Finding optimal locations for new or additional service centers (e.g., Facility Location Problems) [16, 17] is beyond the scope of the present research.

The rest of the paper is organized as follows: Section 2.2 explains the PE-SSTD algorithm. Section 2.3 describes our proposed approaches. We provide correctness proofs of the proposed approaches in Section 2.4. Section 2.5 presents the experimental observations and results. Finally, Section 2.6 concludes the paper.

2.2 PE-SSTD

In this section, we describe the PE-SSTD approach to the CCNVD problem.

2.2.1 Pressure Equalizer Algorithm

The PE-SSTD algorithm starts with a Network Voronoi Diagram (NVD) and iteratively re-assigns graph-nodes until the capacity constraint is met. Recall that an NVD partitions a network into a number of service areas given a spatial network G(N,E,D).

The first core idea in PE-SSTD is to use an NVD as the initial iteration because (1) an NVD represents the optimal sum of the shortest distances under no capacity constraint (Lemma 2) and (2) an NVD creates contiguous service areas (Lemma 1). Thus, by starting with an NVD and keeping changes (re-assignments) to the NVD as minimal as possible, the PE-SSTD algorithm can keep the sum of the shortest distances relatively low and preserve contiguity. In other words, CCNVD can be thought of as capacitating the NVD.

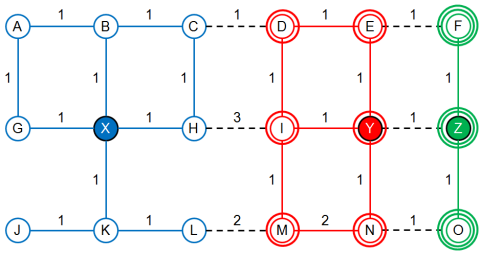
There are two additional reasons to use an NVD as the initial iteration. First, an NVD can be created relatively quickly (see Section 2.4.3), making it a better starting point than, for example, min-cost flow. Second, an NVD is a decent approximation

for natural human behavior in the face of disaster. Recall that after Hurricane Sandy, most people in need of fuel went to their nearest gas stations. In the case of evacuation, people will be inclined to evacuate to the nearest disaster shelter without knowing if the shelter has sufficient capacity available to accommodate the evacuees. Thus, by starting with an NVD and keeping changes it, CCNVD assignment reflects people's natural inclinations, increasing the likelihood of civil compliance with assignments.

The second core idea in PE-SSTD is the Pressure Equalization Graph (PE-Graph), where pressure for a service center s refers to the difference between the capacity of s and the number of nodes allotted to s. Positive values of pressure indicate overload and negative values indicate slack or available capacity. The PE-nodes of a PE-Graph are service centers S in the transportation network. The PE-nodes are of three types: excess, deficit, and balanced. Let $capacity(s)$ be the capacity of a PE-node $s \in S$ and $allotment(s)$ be the number of graph-nodes allotted to s. If $allotment(s) > capacity(s)$, we refer to s as an excess PE-node whose excess value is $allotment(s) - capacity(s)$. On the other hand, if $allotment(s) < capacity(s)$, we refer to s as a deficit PE-node whose deficit value is $capacity(s) - allotment(s)$. We refer to a PE-node s with $allotment(s) = capacity(s)$ as balanced. The collection of graph-nodes allotted to a PE-node s represents the service area (SA) for s. We refer to $SA(s)$ as the service area for s. A PE-edge is inserted from PE-node $s1$ to PE-node $s2$ if any allotted graph-node $n1$ on $s1$ (i.e., $n1 \in SA(s1)$) is connected to any allotted graph-node $n2$ on $s2$ (i.e., $n2 \in SA(s2)$). We refer to both $n1$ and $n2$ as boundary graph-nodes.

Figure 2.5(b) shows the PE-Graph for the NVD of Figure 2.5(a). The graph has three PE-nodes for service centers X, Y, and Z and two PE-edges (i.e., $X \rightarrow Y$ and $Y \rightarrow Z$). PE-node X has an excess of 3 and PE-node Z has a deficit of 3.

The PE-SSTD algorithm tries to satisfy capacity constraints for every service center, maintain service area contiguity constraints, and reduce the sum of the shortest distances from graph-nodes to their allotted service centers (PE-nodes). At each step, the algorithm re-allots a graph-node from an excess PE-node to fulfill the capacity constraint. The effect of re-allotting a graph-node n from $s1$ to $s2$ on an objective function (SA min-sum) can be defined using the following cost function:



(a) Graph(N,E) (NVD) (b) PE-Graph

Node	PE-edge	Cost
C	X → Y	1
H	X → Y	3
L	X → Y	2
E	Y → Z	1
N	Y → Z	1

(c) Boundary Graph-Nodes

Figure 2.5: PE-SSTD: Iteration 1 (Colors show service center allotment)

$$\underset{s1\to s2}{Cost}(a) = \sum_{n\in SA(s1)-\{a\}} sd(n,s1) - \sum_{n\in SA(s1)} sd(n,s1) + \sum_{n\in SA(s2)+\{a\}} sd(n,s2) - \sum_{n\in SA(s2)} sd(n,s2), \tag{2.1}$$

where $sd(n,s)$ is the length of the SA shortest distance from n to s in $SA(s)$ (Lemma 4).

Given an NVD and two service centers $s1$ and $s2$, $ReAllot_{s1\to s2}(n \in SA(s1))$ violates the contiguity constraint in $SA(s2)$ if n is not a boundary graph-node of $SA(s1)$. A key idea behind PE-SSTD is to first choose the best boundary graph-node that minimizes the cost of re-allotment and then re-allot this graph-node to fulfill capacity constraints. Figure 2.5(c) shows boundary nodes for the NVD of Figure 2.5(a). In this example, the best boundary nodes in terms of minimizing the re-allotment cost are node C for $X\to Y$; and node E (or N) for $Y\to Z$.

The PE-SSTD algorithm uses the PE-path that traverses from one excess PE-node to one deficit PE-node on the PE-Graph and re-allots all the best boundary graph-nodes on the PE-path. Challenges during this re-allotment step include maintaining contiguity for SAs. A contiguity checking algorithm is applied to test whether every re-allotment on the PE-path preserves SA contiguity. We refer to this testing as Service Area Contiguity Checking (SACC). This approach smoothly expands (or shrinks) the SAs while preserving SA contiguity.

This process terminates in $O(n)$ iterations, where n is the number of graph-nodes (Lemma 3). In this example, there is one PE-path ($X\to Y\to Z$) to traverse from the excess PE-node X to the deficit PE-node Z. Thus, the algorithm re-allots node C from

X to Y and node E from Y to Z to reduce $excess(X)$ by 1 in the first iteration. This process is repeated two more times to meet capacity constraints by moving node (L,H) from X to Y and node (N,D) from Y to Z.

A large sized network may have many possible PE-paths. The best way to minimize the cost for re-allotment is to find the PE-path that has minimum re-allotment cost across all pairs of excess and deficit PE-nodes. We refer to this as the best PE-path for the current iteration. PE-SSTD invokes a single shortest path algorithm by introducing a super-source and a super-sink node and finds the best PE-path. We first connect the super-source node with all excess PE-nodes and the super-sink node with all deficit PE-nodes. These new connections become PE-edges with a re-allotment cost of 0. Then, one shortest path algorithm on this transformed PE-Graph can identify the pair of excess and deficit PE-nodes with the lowest re-allotment cost.

Algorithm 1 Generalized Pressure Equalizer (PE) Algorithm (Pseudo-code)

Inputs:
- A transportation network ($Graph(N, E)$) with a set of graph-nodes N and edges E.
- A set of PE-nodes (service centers) $S \subset N$ with their capacity C
- Every edge has a distance $d(e)$

Outputs: Capacity Constrained Network Voronoi Diagram ($CCNVD$)

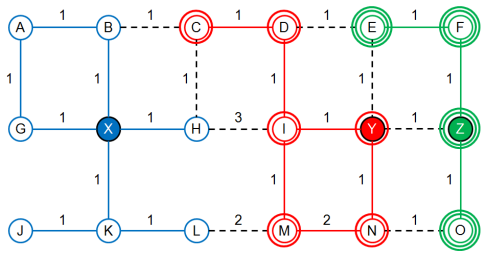
Steps:

1: Create an **initial partition** that preserves service area (SA) contiguity.
2: **while** Any PE-node $s \in S$ has excess graph-nodes **do**
3: Create $PE{-}Graph(S, E_{pe})$ where PE-edge $e_{pe} \in E_{pe}$ connects two adjacent SAs.
4: Find all boundary graph-nodes $N_{bdy} \subset N$ and compute re-allotment cost ($Cost_{s1 \rightarrow s2}(n_{bdy} \in N_{bdy})$).
5: Find the best boundary graph-nodes $N_{best\_bdy} \subset N_{bdy}$ which minimize the re-allotment cost.
6: Group all excess PE-nodes $S_{ex} \subset S$ with a super-source node src_{ex} and group all deficit PE-nodes $S_{df} \subset S$ with a super-sink node $sink_{df}$.
7: Find the best PE-path p in terms of **preserving SA contiguity (i.e., SACC)** as well as minimizing the sum of re-allotment costs from src_{ex} to $sink_{df}$. If no PE-path is founded, then return "no solution found".
8: Re-allot the best boundary graph-nodes (n_{best_bdy}) on the best path p.
9: **end while**
10: return CCNVD. i.e, final allotment of graph-nodes to their service centers.

Algorithm 1 presents the pseudo-code for a generalized version of PE. First, PE creates an initial partition that preserves service area (SA) contiguity (lines 1). In this step, PE-SSTD initializes CCNVD with NVD. It then creates a PE-Graph and finds all boundary graph nodes, as well as the best boundary graph nodes (lines 3-5). After that, it groups excess PE-nodes into a super-source node and groups deficit PE-nodes into

a super-sink node (line 6). Next it searches the PE-graph and finds the best PE-path (line 7). The re-allotments through the best PE-path should preserve SA contiguity. Therefore, the PE-path computation part contains Service Area Contiguity Checking (SACC) (e.g., BFS) to test the contiguity of every service area. If it cannot find the best PE-path that satisfies SA contiguity, then it returns "no solution found". PE then re-allots the best boundary graph-nodes on the best PE-path (line 8). This process continues until the allotment is in line with the capacity of the service centers (line 2). Finally, the updated CCNVD with balanced service centers is returned (line 10).

Figures 2.5–2.8 show the execution of the PE-SSTD algorithm. PE-SSTD starts with NVD (Figure 2.5(a)) and creates a PE-Graph (Figure 2.5(b)). In this example, the service center with an excess is X and the service center with a deficit is Z. PE-SSTD finds the best PE-path to traverse from X to Z (i.e., $X \rightarrow Y \rightarrow Z$) as well as the best boundary graph-nodes adjacent to other SAs. Figure 2.5(c) shows these nodes are C and E.



(a) Iteration 2: Graph(N,E)

(b) PE-Graph

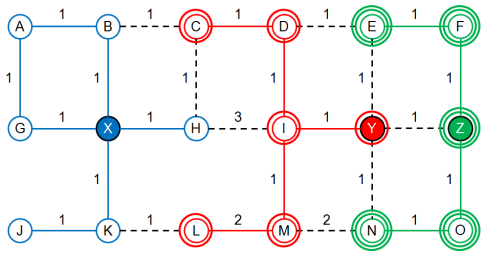
Node	PE-edge	Cost
B	X → Y	3
H	X → Y	3
L	X → Y	2
D	Y → Z	1
N	Y → Z	1

(c) Boundary Graph-Nodes

Figure 2.6: PE-SSTD: Iteration 2

Next, PE-SSTD re-allots the best boundary graph-nodes (C and E) to the service centers in their adjacent SAs (Figure 2.6(a)) and updates the PE-Graph (Figure 2.6(b)). After three iterations, PE-SSTD achieves a balanced allotment (Figure 2.8(b)), having re-allotted nodes L and H from X to Y and nodes N and D from Y to Z, and the algorithm terminates. Figure 2.8(a) shows the resulting Capacity Constrained Network Voronoi Diagram.

After each iteration of PE-SSTD, at least two SAs will have changed due to the transfer of graph-node(s). The SA shortest distance in an affected SA may now be



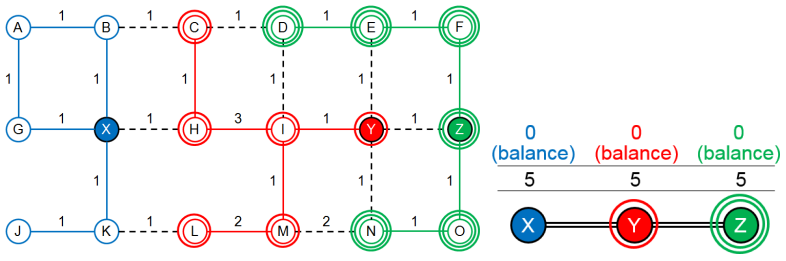
(a) Iteration 3: Graph(N,E)

(b) PE-Graph

Node	PE-edge	Cost
B	X → Y	3
H	X → Y	3
K	X → Y	4
D	Y → Z	1
M	Y → Z	2

(c) Boundary Graph-Nodes

Figure 2.7: PE-SSTD: Iteration 3



(a) Iteration 4: Graph(N,E)

(b) PE-Graph

Figure 2.8: PE-SSTD: Iteration 4 produces a CCNVD

different due to the addition or removal of graph-node(s). As a result, an optimal algorithm would have to run multiple shortest path computations after each iteration to ensure that it is always choosing the best node for transfer. This is very computationally expensive, especially for large networks where many iterations may be needed. One possible approach to reduce computation time (albeit at the expense of solution quality) is to use an approximate cost function as defined below.

$$\underset{s1 \to s2}{\hat{Cost}}(a) = nd(a, s2) - nd(a, s1), \tag{2.2}$$

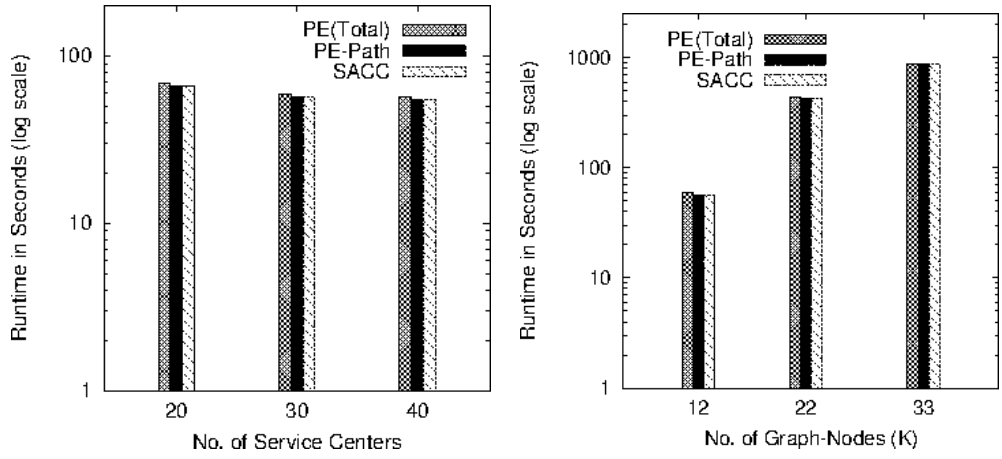
where $nd(a, s)$ is the length of the network shortest distance from a to s in G.

Algorithms that use this approximation may choose non-ideal nodes for transfer, but will never need to recompute shortest paths after the first iteration since the approximate distance (i.e., network shortest distance) does not change even when the SA changes. In our approach, PE-SSTD uses the approximate cost function to reduce the computational cost.

2.2.2 Limitation of PE-SSTD

In this subsection, we demonstrate that the bottleneck point of PE-SSTD is Service Area Contiguity Checking (SACC). First, we used a Miami road map consisting of $12,402$ nodes and varied the number of service centers. Then we fixed the number of service centers (e.g., $|S|$) to 30 and varied the number of graph-nodes (e.g., $|N|$) using the five road maps, as shown in Table 2.4. We used BFS for the SACC component. The PE-Path computation part of PE-SSTD contained both SACC (i.e., BFS) and the best path computation. Service center locations were randomly selected and execution times were averaged over 50 test runs. Figure 2.9 shows the results of bottleneck analysis. As expected, SACC is the main bottleneck in PE-SSTD in terms of computational cost. Reminder that the time cost of SACC by BFS is propositional to the number of edges in the worst case. As the number of service centers increases, the gap between the total cost of PE-SSTD and SACC slightly increases (Figure 2.9(a)). This is because the size of SAs becomes smaller as the number of service centers grows. Although the increase is slight, it nevertheless suggests that BFS takes less time due to smaller SA size to be traversed. To verify this interpretation, we fixed the number of service centers and increased the size of the road maps. As the number of nodes increases, the gap between the total cost of PE-SSTD and SACC decreases (Figure 2.9(b)). This is because the

size of SAs becomes larger with increasing number of nodes on the road map. In any case, we conclude that the main bottleneck point of PE-SSTD is SACC.



(a) Effect of Number of Service Centers ($|N|=12,402$)

(b) Effect of Number of Nodes ($|S|=30$)

Figure 2.9: Run-time bottleneck analysis of PE-SSTD algorithm

2.2.3 Proof and analysis

In this section, we prove that the PE-SSTD is correct, i.e., the PE-SSTD algorithm creates a CCNVD.

Lemma 3. *The PE-SSTD algorithm terminates after at most n iterations, where n is the number of graph-nodes*

Proof. Each iteration reduces the number of allotted graph-nodes for excess service centers by one and increases the number of allotted graph-nodes for deficit service centers by one. The maximum possible number of allotted graph-nodes for excess service centers is n. Therefore, the maximum iteration is at most n. □

Lemma 4. *Given a graph-node a and two service centers s1 and s2, the cost for* $ReAllot_{s1\to s2}(a \in SA(s1))$ *is defined by* $Cost_{s1\to s2}(a) = \sum_{n\in SA(s1)-\{a\}} sd(n,s1) - \sum_{n\in SA(s1)} sd(n,s1) + \sum_{n\in SA(s2)+\{a\}} sd(n,s2) - \sum_{n\in SA(s2)} sd(n,s2)$*, where* $sd(n,s)$ *is the length of the SA shortest path distance from n to s.*

Proof. The sum of the SA shortest distances for two service centers $s1$ and $s2$ is $A = \sum_{n \in SA(s1)} sd(n, s1) + \sum_{n \in SA(s2)} sd(n, s2)$. After re-allotting a from $s1$ to $s2$, the sum of the SA shortest distances becomes $B = \sum_{n \in SA(s1)-\{a\}} sd(n, s1) + \sum_{n \in SA(s2)+\{a\}} sd(n, s2)$. Therefore, the increased sum of the SA shortest distances (cost) is defined by $B - A$. □

Lemma 5. *The normal termination of PE-SSTD meets capacity and contiguity constraints.*

Proof. At termination, no excess PE-node exists on the PE-Graph (Lemma 3). Since each re-allotment satisfies SA contiguity constraints, PE-SSTD meets capacity and contiguity constraints at termination. □

2.3 Proposed Algorithms

In this section, we introduce two novel approaches, namely (a) Block Tree Contiguity Checking (BTCC) and (b) Graph Minor. BTCC is used to reduce the computational cost of testing for service area (SA) contiguity (Line 7 in Algorithm 1). Graph Minor is used to group a set of graph-nodes and create a novel initial partition (Line 1 in Algorithm 1) to speed up the re-allotment.

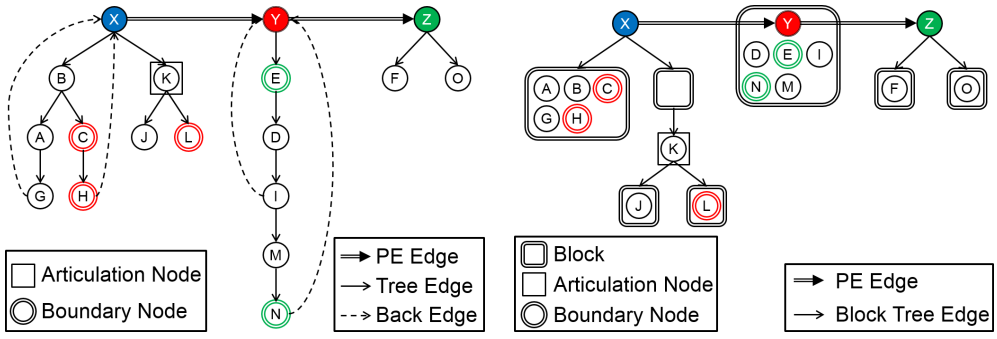
2.3.1 BTCC approach to SA contiguity

The main performance bottleneck of PE-SSTD is the check for SA contiguity. In this section, we introduce our new method to reduce the computational cost of this step. In formal terms, the Service Area Contiguity Checking (SACC) problem is defined as follows: given a connected graph SA, test whether SA contiguity is preserved after insertion of a node $n_{ins} \notin SA$ and removal of a node $n_{rem} \in SA$. In a naive approach, we may use graph traversal algorithms (e.g., DFS or BFS), but the computational cost of these algorithms is linear in the number of edges. This high running time for SACC makes it hard for PE-SSTD to handle large sized transportation networks because it may have to extensively search SAs several times, in each iteration.

Consider the example in Figure 2.7. Service area X has three boundary nodes (i.e., B, H, and K) to service area Y and service area Y has two boundary nodes (i.e., D and M) to service area Z. Either boundary node B, H, or K in service area X can be moved

into service area Y, but no boundary node in service area Y can be moved into service area Z without violating the SA contiguity constraint. Assume that the PE algorithm first moves boundary node H to service area Y. After node H's insertion, we can move boundary node D into service area Z without violating SA contiguity. However, naive graph traversal algorithms must search all of service area Y several times to figure out which of its boundary nodes are movable to service area Z.

To improve the efficiency of SA contiguity checking, we employ a novel tree structure based on the following idea. A connected graph may contain a node whose removal disconnects the remaining nodes. We refer to this node as an articulation node [21]. Since a SA is a connected graph, it may contain articulation nodes. According to Tarjan's algorithm [22], we can create a DFS-spanning tree in linear time and detect these articulation nodes in constant time. A graph with no articulation nodes is called bi-connected or non-separable. A maximal bi-connected sub-graph of a graph is called a block [21]. Since our main focus is finding articulation nodes in the SA, we group these non-articulation nodes into blocks to simplify the representation of the DFS-spanning tree. We refer to this tree as a block tree [21, 22]. Our approach uses a simpler block tree that consists only of blocks and articulation nodes.



(a) DFS-spanning tree (b) Block tree

Figure 2.10: Example of DFS-spanning and block tree: Iteration 1

Figure 2.10 shows examples of DFS-spanning and block trees generated from Figure 2.5(a). The edges of the original graph can be divided into two types: tree edges and back edges. A tree edge belongs to the DFS-spanning tree itself; it connects a node to one of its descendants whereas a back edge connects a node to one of its ancestors.

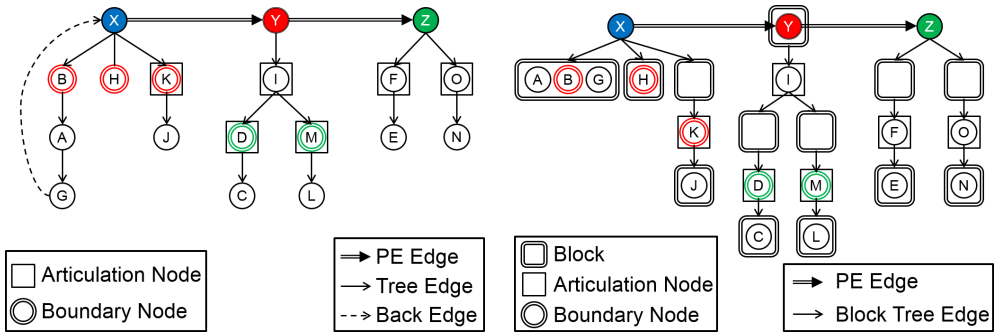
Figure 2.11(a) shows tree edges (solid lines) and back edges (dotted lines) based on the original graph.

After creating the DFS-spanning tree, we can easily see that every leaf node is a non-articulation node because its removal does not separate the SA. Consider the case of a non-leaf node that has children. If every child has a path to an ancestor of the non-leaf node with tree or back edges, then the non-leaf node is not an articulation node because these paths create a bi-connected sub-graph that includes the non-leaf node and its children. However, if a child has no path to any ancestor of the non-leaf node, it becomes an articulation node because this child becomes orphaned from the root after removal of its parent node. In Figure 2.10(a), node K is a non-leaf node and has two children, J and L. Since neither child has a path to an ancestor of node K, node K removal separates the SA into three connected graphs (e.g., (A, B, C, G, H), (J), and (L)). However non-leaf node B has two children, both of whom have a path to an ancestor of node B (e.g., $A \rightarrow G \rightarrow X$ and $C \rightarrow H \rightarrow X$). Therefore, node B is not an articulation node and its removal does not separate the SA. We then group these non-articulation nodes into blocks and get a block-tree shown in Figure 2.10(b).

A block tree representation simplifies SA contiguity checking because it allows us to determine whether a node in the SA is an articulation or not in constant time. However, this approach has limited ability to handle node updates (e.g., node insertion and deletion). PE re-allots one graph-node to its adjacent SA along the best PE-path. During computation of the best PE-path, a new block tree needs to be created and maintained every time a graph-node is re-allotted to its adjacent SA. In the block tree in Figure 2.11(b) (generated from Figure 2.7(a)), service area Y has two boundary graph-nodes, D and M, both of which are articulation nodes. However, they may no longer be articulation nodes after H is inserted into the service area Y.

After inserting graph-node H into service area Y, we have to create a new block tree to see which graph-nodes in Y, if any, are articulation nodes under the new condition. Since this maintenance takes more than constant time, it will be a bottleneck point in each iteration of the PE algorithm. Instead of block-tree maintenance, what we need is a quick way to test if a graph-node in the SA is an articulation node after a single graph-node insertion. Our proposed approach achieves this by exploiting look-up tables and the block tree structure. In the next subsection, we describe a novel solution to the

SACC problem that checks SA contiguity under single graph-node insertion and deletion in the SA so that we can decide which graph-nodes are movable on the PE-path.



(a) DFS-spanning tree (b) Block tree

Figure 2.11: Example of DFS-spanning and block tree: Iteration 3

Block Tree Contiguity Checking Algorithm

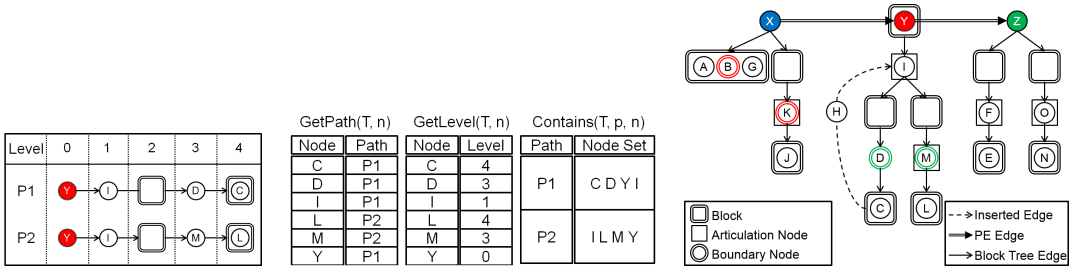
The Block Tree Contiguity Checking (BTCC) algorithm is a novel method to optimize the computational time cost of SA contiguity checking after linear pre-processing. For the remainder of our discussion, we assume that a node in the SA has a constant degree. In a transportation network, this special case is adequate because the highest degree of any node in a highway network is approximately 4 [23]. The following is a list of the main function used by BTCC to analyze the block tree structure.

- **GetPath**(T, n): Given a block tree T, a node n, and paths from the root to leaf nodes, return a path that contains node n.
- **GetLevel**(T, n): Given a block tree T and a node n, return a level of node n on the block tree.
- **Contains**(T, p, n): Given a block tree T, a path p, and a node n, return true if the path p contains the node n.
- **GetLCA**$(T, n_1, n_2, \ldots, n_k)$: Given a block tree T and k nodes $n_1, n_2, \ldots, n_k$, return the lowest common ancestor (LCA) of $n_1, n_2, \ldots, n_k$ that is located farthest from the root (i.e., the node at the highest level of T).
- **BTCC**(T, n_{ins}, n_{rem}): Given a block tree T, a node $n_{ins} \notin T$, and a node $n_{rem} \in T$, return true if node n_{rem} is an articulation node after node n_{ins} has been inserted into

T.

The BTCC algorithm proceeds in three steps. First, it creates a block tree according to the DFS algorithm [22]. Then it creates look-up tables for three functions: $GetPath()$, $GetLevel()$, and $Contains()$. For the $GetLCA()$ function, we use the LCA algorithm [24, 25, 26, 27], which can answer in constant time after linear pre-processing of the block tree T. Finally, it calls $BTCC(T, n_{ins}, n_{rem})$ to see if node n_{rem} is an articulation node now that node n_{ins} has been inserted into T.

Consider again the block tree in Figure 2.11(b). Service area Y has two boundary nodes (e.g., D and M), both of which are articulation nodes. We first create a path table that contains all paths from the root to its leaf nodes (Figure 2.12(a)). Next, we create look-up tables that return the value for three functions, $GetPath()$, $GetLevel()$, and $Contains()$ (Figure 2.12(b)). We also need to create a look-up table for $GetLCA()$ according to the LCA algorithm [25, 26, 27]. We assume that node H is moving from service area X to Y (Figure 2.12(c)). When it does, node D is no longer an articulation node because insertion of H has created a bi-connected sub-graph (e.g., C, D, H, and I). We now show how to prove that node D is not an articulation node in constant time.



(a) Path table for service area Y in block tree: Iteration 3

(b) HashTables

(c) Block graph after moving graph-node H to service center Y

Figure 2.12: Example of Path table and block graph: Iteration 3

As we mentioned before, node D is an articulation node in the block tree for service area Y, and node H is a node inserted into service area Y (Figure 2.12(c)). There are two incident nodes of H (i.e., C and I) and the lowest common ancestor (LCA) of these two nodes is I. Let P be a set of paths that connects the lowest common ancestor and incident nodes of n_{ins}. If P covers all children of n_{rem}, then node n_{rem} is not an

articulation node because the insertion of node n_{ins} creates a bi-connected sub-graph that contains node n_{rem}. In this example, node D has only one child (C) and path $P1$ covers node C. Because the level of the lowest common ancestor (I) is 1 and the level of C is 4, the sub-path between I and C of $P1$ also covers node C. Therefore, node D is no longer an articulation node. This is tested simply by successively calling the predefined functions (e.g., $GetPath()$, $GetLevel()$, $Contains()$, and $GetLCA()$).

Our approach creates look-up tables for these predefined operations and uses a hash function that offers constant time performance for the basic operations (e.g., add and search) [28, 29].

Data Structures

- **GetLevel**(T, n): After computing BFS, we store levels of nodes $n \in T$ and create a look-up table.
- **GetPath**(T, n): After computing BFS, we create paths from the root to leaf nodes on the block tree T. The number of paths is bounded by the number of leaf nodes. Every node $n \in T$ is associated with one path and inserted into a look-up table. If more than one path contains the node, then the shortest path is selected.
- **Contains**(T, p, n): After creating paths from the root to leaf nodes on block tree T, every path p is associated with a set of nodes $n \in p$ and inserted into a look-up table.

At the beginning of each iteration, the PE algorithm constructs look-up tables for these three functions which serve as an input of the BTCC algorithm. These look-up tables require linear-time pre-processing (e.g., BFS) and provide a constant time look-up operation. As mentioned previously, we use the LCA algorithm for the $GetLCA()$ operation [25, 26, 27] and insert the pre-computed answers into a look-up table after linear pre-processing of the block tree [21].

Algorithm 2 presents the pseudo-code for the BTCC algorithm. First, BTCC checks whether node n_{ins} has only one incident on the block tree and whether the incident is the node n_{rem} (lines 1-2). If this is true, then node n_{rem} becomes an articulation node (Lemma 6). Next, it checks whether node n_{rem} is an articulation node on the block tree (line 3). If it is not, it is also not an articulation node after insertion of node n_{ins} (Lemma 7). BTCC then finds all incident nodes of n_{ins} and children of n_{rem} as well as the lowest common ancestor of $n_{lca\_incdt\_ins}$ (lines 4-5). After that, it simply checks if the level of $n_{lca\_incdt\_ins}$ is greater than that of n_{rem}. If it is not, it returns true because

Algorithm 2 Block Tree Contiguity Checking (BTCC) Algorithm (Pseudo-code)

Inputs:
- A block tree $T(N_{art}, N_{blk}, E)$ with a set of articulation nodes N_{art}, blocks N_{blk} and edges E.
- A node $n_{ins} \notin T$ that will be inserted into T.
- A node $n_{rem} \in T$ that will be removed from T.
- Look-up tables: $GetLevel()$, $GetPath()$, $Contains()$, $GetLCA()$.

Outputs: Return true if n_{rem} is an articulation node after inserting n_{ins} into T

Steps:

1: **if** n_{ins} has only one incident in T and the incident is n_{rem}
2: **then** return true.
3: **if** n_{rem} is not an articulation node in T **then** return false.
4: $N_{incdt\_ins} \leftarrow$ (incident nodes of n_{ins}) $\in T$
5: $N_{ch\_rem} \leftarrow$ children of n_{rem}, $n_{lca\_incdt\_ins} \leftarrow LCA(N_{incdt\_ins})$
6: **if** $level(n_{lca\_incdt\_ins}) \leq level(n_{rem})$ **then** return true.
7: $P_{lca \rightarrow incdt} \leftarrow$ paths from $n_{lca\_incdt\_ins}$ to $n_{incdt\_ins} \in N_{incdt\_ins}$
8: **if** $P_{lca \rightarrow incdt}$ cover all $n_{ch\_rem} \in N_{ch\_rem}$ **then** return false.
9: return true.

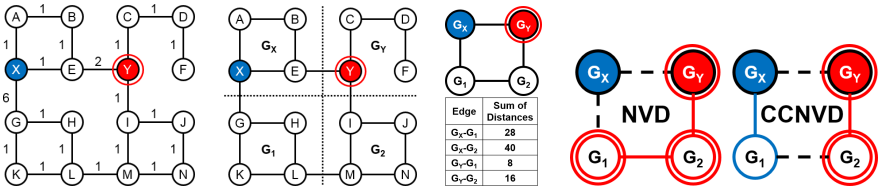
no children have a path to an ancestor of n_{rem} (lines 6). Next, it checks if these children (e.g., n_{ch_rem}) can be covered by the path from $n_{lca\_incdt\_ins}$ to all $n_{incdt\_ins} \in N_{incdt\_ins}$. If they can, it returns false (lines 7-8). Finally, it returns true after passing all the above criteria (line 9).

The BTCC approach does not require maintaining bi-connectivity [30, 31]. The SACC problem is a special case of a dynamic connectivity problem where only a boundary node can be inserted into SA or removed from SA [32]. Simply, after linear pre-processing, BTCC examines all boundary nodes and tests if the graph is connected after one node insertion (n_{ins}) and one node removal (n_{rem}).

2.3.2 Graph Minor approach to initial partition

Although BTCC reduces the computational cost for the bottleneck of the PE-SSTD algorithm, it may be inapplicable for sizable road networks (e.g., USA road map). Thus we propose a more scalable algorithm using a Graph Minor. In graph theory, a minor of graph G can be formed from G by contracting edges and nodes [21]. Our approach uses a Graph Minor to group a set of graph-nodes and move multiple graph-nodes instead of one graph-node. As a pre-processing step, we use a balanced min-cut graph partitioning to decompose network G into connected components $c \in C$ and create a minor of G by contracting the edges and nodes in every $c \in C$. There are two reasons to use this method. First, a balanced min-cut partitions the network into same size

sub-networks, making it possible to create minor nodes of the same load. Second, since min-cut minimizes the number of edges between partitions, it can easily create a set of connected graph-nodes for every minor-node.



(a) Input (b) Min-cut partitioning (c) Graph Minor (d) Minor-NVD & Minor-CCNVD

(e) CCNVD

Figure 2.13: Example of PE algorithm using Graph Minor

Figure 2.13 illustrates constructing of a CCNVD using Graph Minor. The input is a transportation network (14 graph-nodes $(A, B, \ldots, N)$ and two service centers (X and Y)). Figure 2.13(b) illustrates balanced min-cut partitioning of the network. After partitioning, minor-nodes for the service centers are chosen (e.g., G_x and G_y). If more than one service center is located in the same minor-node, the set of graph-nodes in the minor-node is partitioned again with the NVD algorithm. Figure 2.13(c) shows a minor of the network after node and edge are contracted in every partition. As shown in the accompanying table, every minor-edge is associated with the sum of the shortest distances from a set of nodes in a partition to a service center. Next, the Minor-NVD and Minor-CCNVD are created with the PE algorithm. Finally, Figure 2.13(e) shows the CCNVD after expanding of every minor node.

The key idea behind the PE-Minor approach is to reduce the size of the network and move a set of graph-nodes through the PE-path. The manageable size of Graph Minor makes it theoretically possible to develop efficient CCNVD methods (see Section 2.4.3).

2.4 Analysis on the quality of the proposed approaches

In this section, we prove that PE with BTCC (or Graph Minor) approach creates CCNVD.

2.4.1 Analysis of BTCC approach to SA contiguity

The following lemmas prove the correctness of the Block Tree Contiguity Checking (BTCC) algorithm.

Lemma 6. *If a node n_{ins} inserted into a SA has only one incident in the SA and the incident is a node n_{incdt_ins}, then node n_{incdt_ins} becomes an articulation node.*

Proof. After a node n_{ins} is inserted into the SA, the removal of n_{incdt_ins} disconnects the node n_{ins} from the SA. Therefore, node n_{incdt_ins} is an articulation node. □

Lemma 7. *Except in the case of Lemma 6, given a SA, if a node n is not an articulation node in the SA, then it is also not an articulation after insertion of a node n_{ins} into the SA.*

Proof. If a node n is not an articulation, the node is a part of a bi-connected sub-graph. Since the insertion of node n_{ins} does not decrease the connectivity of the SA, node n is not an articulation node. □

Lemma 8. *Given a block tree T, an inserted node ($n_{ins} \notin T$), a set of its incident nodes ($n_1, n_2, \ldots, n_k \in T$), and a maximum node degree (n_{maxdeg}), $GetLCA(T, n_1, n_2, \ldots, n_k)$ gives an answer in $O(n_{maxdeg})$ time after linear-time pre-processing.*

Proof. Given two nodes u and v, the Lowest Common Ancestor (LCA) algorithm gives an answer in constant time after linear-time pre-processing [28, 29]. The LCA function has commutative (e.g., $LCA(u, v) = LCA(v, u)$) and associative (e.g., $LCA(LCA(u, v), w) = LCA(u, LCA(v, w))$) properties. The number of incident nodes (e.g., k) is bounded by the maximum node degree (n_{maxdeg}). Therefore, $GetLCA(T, n_1, n_2, \ldots, n_k)$ can be computed with a time cost of $O(n_{maxdeg})$ by calling $LCA(\ldots LCA(LCA(n_1, n_2), n_3) \ldots, n_k)$. □

Lemma 9. *After inserting a node n_{ins} and its incident edges E_{ins} into a block tree, an articulation node n is no longer an articulation node if its children have a path to one of the ancestors of node n.*

Proof. Assume that after a node and its incident edges are added into the block tree, node n has children that have a path to one of its ancestors of node n with these new added edges E_{ins}. Then, even with removal of node n, its children maintain a connection with one ancestor of node n with edges E_{ins}. Therefore, node n is not an articulation node because its removal does not separate the graph. □

Lemma 10. *The BTCC algorithm is correct and does not affect the solution quality.*

Proof. The BTCC algorithm finds articulation node correctly and completely. Based on the definition of an articulation node [21], BTCC produces exactly the same results as the naive traversal algorithms (e.g., BFS and DFS). □

2.4.2 Analysis of Graph Minor approach to initial partition

The following lemmas prove the CCNVD with PE-Minor satisfies both capacity and SA contiguity constraints.

Lemma 11. *Given a minor $H(N_m, E_m)$ of $G(N, E)$, where every minor-node ($n_m \in N_m$) is created by contracting the same size connected sub-graph ($N_s \subset N$), the CCNVD of H satisfies both capacity and SA contiguity constraints of G.*

Proof. The CCNVD of H is a set of same size connected sub-graph of H. Let this sub-graph be $SA_H(s \in S)$. Since every minor-node ($n_m \in N_m$) is the same size connected sub-graph of G, every expansion of $SA_H(s)$ is the same size connected sub-graph of G. □

2.4.3 Algebraic Cost Model of the PE Algorithm

We developed a cost model for the proposed PE algorithms to estimate their computational cost. Let n be the number of graph-nodes, let k be the number of service centers, let m be the number of edges, and let n_{maxdeg} be the maximum node degree. Assume that n_{maxdeg} is sufficiently small or constant (e.g, transportation network). Since

$m = O(n \times n_{maxdeg})$, $m = O(n)$. All PE approaches create an NVD as an initial solution at a cost of $O(k \cdot n \cdot \log n)$ using a reversed Dijkstra's algorithm [33].

PE-SSTD

First, PE-SSTD scans all edges and finds all boundary graph-nodes. This step takes $O(m)$. At each iteration in the best PE path computation, the algorithm extracts a PE-node s from the Fibonacci heap. This takes $O(\log k)$. Assume that PE visited another PE-node s_{pre} just before visiting the current PE-node s on the PE path and that the best boundary graph-node n_{pre_rem} is associated with PE-node s_{pre}. Then PE-SSTD chooses all boundary graph-nodes (n_{rem}) in the $SA(s)$ and checks the SA contiguity for the $SA(s)$ under the insertion of node n_{pre_rem} and the removal of node n_{rem}. Since the total number of boundary edges in all SAs is $O(m)$ and SACC with BFS takes $O(m)$, the operation to find the best boundary graph-nodes takes $O(m^2)$ overall. After finding the best boundary graph-node n_{rem} as well as its adjacent PE-node s_{next}, PE inserts s_{next} into the Fibonacci heap. The total number of insertions is $O(k)$. Since a PE-graph is a sparse graph, the total number of decrease-keys is $O(k)$. The number of best PE paths is bounded by n according to Lemma 3. Therefore, the cost model is $O(n \cdot (m + m^2 + k \cdot \log k))$. Assume that $k \cdot \log k << m^2$ and n_{maxdeg} is sufficiently small or constant. Then the complexity is $O(n^3)$.

BTCC

The main difference with the previous approach is Service Area Contiguity Checking by BTCC. First, PE with BTCC creates a block tree for all SAs at a cost of $O(m)$. Since service area contiguity checking with BTCC takes $O(n_{maxdeg})$ time (Lemma 8), the operation to find the best boundary graph-nodes takes $O(m \cdot n_{maxdeg})$ overall. Therefore, the cost model is $O(n \cdot (m + m \cdot n_{maxdeg} + k \cdot \log k))$. Assume that n_{maxdeg} is sufficiently small or constant. Then the complexity is $O(n^2)$.

Graph-Minor

Reprocessing for a Graph Minor reduces the size of the network. The basic idea of the Graph Minor approach is to choose a number k and partition the given network into

n/k sub-networks of size k. For efficiency, we fixed k to $\sqrt[4]{n}$, which grows slowly as the size of network increases. Therefore, we can reduce n nodes to $\sqrt[4]{n^3}$ minor-nodes. If PE with BTCC is applied to Graph Minor, then the complexity of the PE algorithm is $O(n \cdot \sqrt{n})$.

Table 2.2: Algebraic Comparison Of Computational Cost

Algorithm	Computational Cost		
	SACC	# iterations of PE	Total
min-cost flow [34]	-	-	$O(n^2 \cdot \log n)$
PE-SSTD	$O(m)$	$O(n)$	$O(n^3)$
PE w/ BTCC	$O(n_{maxdeg})$	$O(n)$	$O(n^2)$
PE w/ BTCC & Minor	$O(n_{maxdeg})$	$O(\sqrt[4]{n^3})$	$O(n \cdot \sqrt{n})$

Table 2.2 shows computational costs for the PE approaches and min-cost flow. The cost model is analyzed based on two design decisions: Service Area Contiguity Checking (SACC) and initial partition.

2.4.4 Space Complexity of the PE Algorithms

Assume that n_{maxdeg} is sufficiently small or constant. Then the space complexity of the input graph is $O(n)$. All PE approaches require $O(k \cdot n)$ space to store distances from every graph-node to every service center.

BFS requires $O(n)$ space to check SA contiguity. Therefore, PE-SSTD takes $O(k \cdot n)$. BTCC requires $O(n)$ to create tables for $GetLevel()$, $GetPath()$, and $GetLCA()$. In addition, BTCC requires $O(n^2)$ to create a table for $Contains()$ because the number of paths is $O(n)$ and the size of the node set is $O(n)$ (see Figure 2.12(b)). Therefore, PE with BTCC takes $O(n^2 + k \cdot n)$. Graph-Minor reduces the number of nodes from n to $\sqrt[4]{n^3}$. BTCC with Graph-Minor requires $\sqrt[2]{n^3}$. Therefore, PE with Graph-Minor takes $O(\sqrt[2]{n^3} + k \cdot n)$.

2.5 Experimental Evaluation

We conducted experiments to evaluate performance of the PE algorithms. The overall goal was to show the performance improvements to create CCNVD that can be obtained

by the PE algorithm with BTCC and Graph Minor. We wanted to answer four questions: (1) What is the effect of the number of service centers? (2) What is the effect of the size of the network (i.e., number of graph-nodes)? (3) Is the BTCC algorithm correct and is solution quality preserved? (4) How often is the capacity constraint (or contiguity constraint) violated by NVD (or MCF) ? (5) Is the PE algorithm scalable?

2.5.1 Experiment Layout

Table 2.3 shows candidates of the PE algorithm. PE algorithms is classified with two design decisions: Service Area Contiguity Checking (SACC) and initial contiguous service areas.

Table 2.3: Candidates of PE algorithm

Design Decision / Candidate	initial contiguous SAs	SACC
PE-SSTD	NVD with Graph	BFS
PE-BTCC	NVD with Graph	BTCC
PE-Minor	NVD with Graph Minor	BTCC

Figure 2.14 shows our experimental setup. We chose five different municipal areas in the U.S. from OpenStreetMap [35]. Table 2.4 shows the number of nodes (e.g., $|N|$) and edges (e.g., $|E|$) on the chosen areas. We also retrieved the locations of service centers (e.g., schools, gas stations, or shelters) from OpenStreetMap. In the analysis, we fixed the number of service centers (e.g., $|S|$) and randomly chose their locations from each map. Then we created a Capacity Constrained Network Voronoi Diagram (CCNVD). For simplicity, we assumed that all service centers had the same capacity and that the service centers together could serve all the people allowed during the time intervals of interest.

Ideally we would test our proposed family of algorithms against comparable algorithms from related work. Unfortunately, we found no algorithms in the literature that honor both capacity and contiguity constraints as detailed in Section 2.5.3 which we describe as follows. The closest algorithm may meet one of these constraints. For example, MCF meets capacity constraint, but not SA contiguity constraint. We include MFC in our experiment to serve as a loose lower bound solution quality as it does not

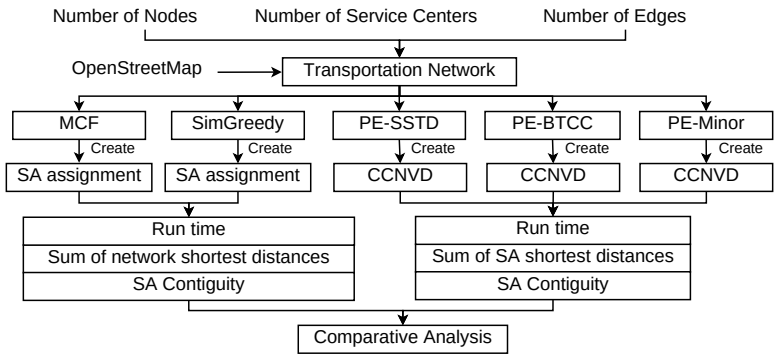


Figure 2.14: Experiment Layout

Table 2.4: Transportation Networks (Source: OpenStreetMap)

Area	No. of Nodes ($\|N\|$)	No. of Edges ($\|E\|$)
Miami, FL	12,402	40,074
Boston, MA	22,278	66,200
New Orleans, LA	32,744	105,008
Twin Cities, MN	56,168	179,162
Chicago, IL	121,042	386,916

consider SA contiguity. We also include a straightforward alternative to evaluate the effectiveness and efficiency of the proposed approaches, as shown in the following.

Simplistic greedy approach (SimGreedy): Given a network G and a set of service centers S, we begin with empty Service Areas. In each iteration, SimGreedy tries to find graph-node x which is nearest to one of the deficit service centers $s \in S$ and is directly connected to $SA(s)$. If SimGreedy finds such a graph-node, it assigns it to the corresponding nearest service center s. If it does not, then it ignores the SA contiguity constraint and assigns the graph-node to the corresponding nearest service center s. The computational cost of SimGreedy is $O(k \cdot n \cdot \log n)$ due to the shortest distance computation from graph-nodes to service centers. SimGreedy requires $O(k \cdot n)$ space to store distances from graph-nodes to service centers.

We tested five approaches: (1) min-cost flow (MCF) (2) SimGreedy (3) PE-SSTD, (4) PE-BTCC, (5) PE-Minor.

As a pre-processing for Graph Minor, we used Metis [36] to equitably partition the network. In our analysis, we found that Metis could not create perfect balanced

Table 2.5: List of Approaches

Approach	Description
MCF	Minimum Cost Flow for Assignment Problem
SimGreedy	Simplistic Greedy Approach for Assignment Problem
PE-SSTD	PE algorithm that uses BFS for SA contiguity checking
PE-BTCC	PE algorithm that uses BTCC for SA contiguity checking
PE-Minor	PE-BTCC that uses Graph Minor for initial partition

partitions under contiguity constraints. Since Metis allows imbalanced partitions, we ran the PE-BTCC after PE-Minor if the partitions were imbalanced.

We evaluated these algorithms by comparing the impact on performance of (1) number of service centers and (2) size of the transportation network. The algorithms were implemented in Java 1.7 with a 8 GB memory run-time environment. All experiments were performed on an Intel Xeon CPU E5472 machine running Ubuntu 10.04.4 LTS with 8 GB of RAM.

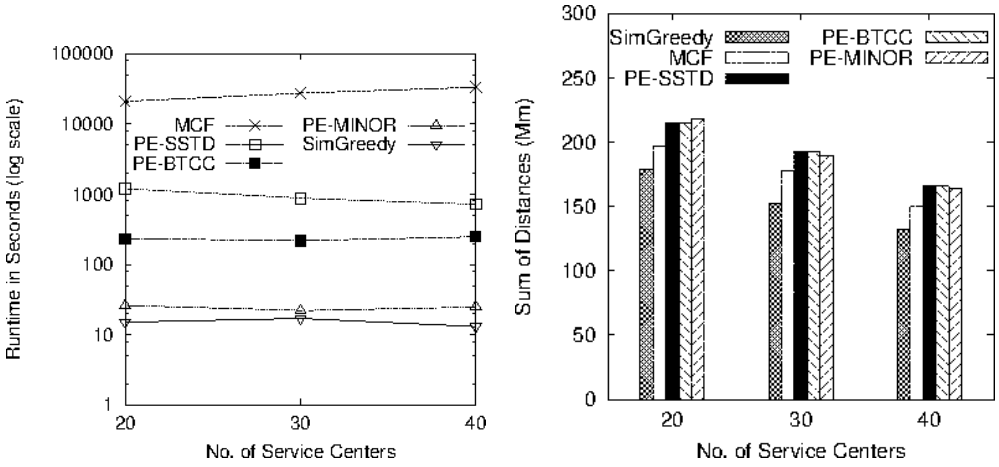
2.5.2 Experiment Results and Analysis

Effect of the number of service centers

The first experiment evaluated the effect of the number of service centers on the performance of the algorithms. Performance measurements were execution time and the sum of the SA shortest distances (SA min-sum). We used a New Orleans road map consisting of $32,744$ nodes. The number of service centers was varied from 20 to 40. The locations of service centers were randomly chosen 50 times. Figures 2.15(a) gives the execution times. As can be seen, PE-BTCC outperforms PE-SSTD. The performance gap decreases as the number of service centers increases. This is because the size of the SA decreases as the number of service centers increases. PE-Minor outperforms other approaches because Graph Minor reduces the size of the network and speeds up the PE algorithm. When comparing the sum of the SA shortest distances (Figure 2.15(b)), we see that both PE-BTCC and PE-SSTD perform identically. This means that PE-BTCC has no effect on the solution quality. PE-Minor performs almost identically to PE-BTCC. The performance of both MCF and SimGreedy was measured by min-sum instead of SA min-sum because the two approaches violated contiguity constraint in

every test run. SimpGreedy is faster than MCF, but MCF is better than SimGreedy in terms of min-sum. As the number of service centers increases, the sum of the SA shortest distances decreases.

All PE approaches are faster than MCF because they consider only boundary nodes on PE-Path for the re-allotment unlike MCF which explores all nodes in each iteration. In addition, BTCC data-structure significantly reduces the computational cost of PE due to lower cost of SA continuity checking. PE with Graph-Minor is faster than others because it reduces the size of the input-graph and re-allots a set of nodes through the PE-Path. Both MCF and SimGreedy fail to create a CCNVD because they have no notion of SA contiguity.



(a) Run-time Comparison

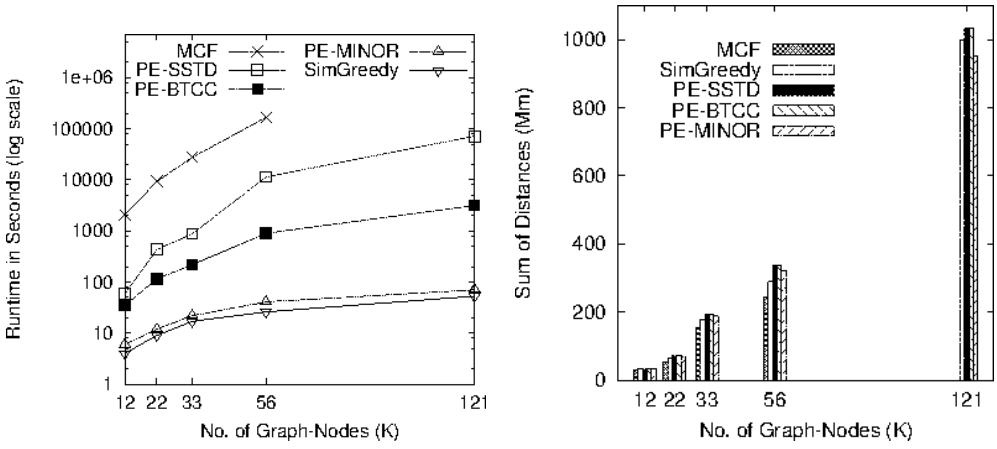
(b) Comparison of Sum of SA shortest distances

Figure 2.15: Effect of the number of service centers ($|N| = 32,744$) (MCF and SimGreedy use min-sum and others use SA min-sum)

Effect of network size

The second set of experiment evaluated the effect of the network size on algorithm performance. We increased network size using the five road maps listed in Table 2.4. We fixed the number of service centers to 30 and incrementally increased the number of graph-nodes from $12,402$ to $121,042$. Service center locations were chosen randomly and execution times were averaged over 50 test runs for each road network. Since MCF

was not scalable for large sized network datasets, we skipped the test for the Chicago road map (i.e., 121, 042 nodes). Figure 2.16(a) shows that PE-BTCC significantly outperforms PE-SSTD and PE-Minor outperforms other approaches. As the number of graph-nodes increases, so does the performance gap. This is because the size of the SA increases as the size of the network increases. Figure 2.16(b) shows that PE-BTCC performs exactly the same as PE-SSTD. PE-Minor shows better solution quality when the size of the network increases. This is because a minor-node groups a set of graph-nodes and approximately preserves the value of the sum of the SA shortest distances. As the number of graph-nodes increases, the sum of the SA shortest distances also increases. The results of the experiments show that PE-BTCC is faster than PE-SSTD with no loss of solution quality and that PE-Minor is a practical choice when the size of the network is very large. Both MCF and SimGreedy violated the contiguity constraint in every test run.



(a) Run-time Comparison

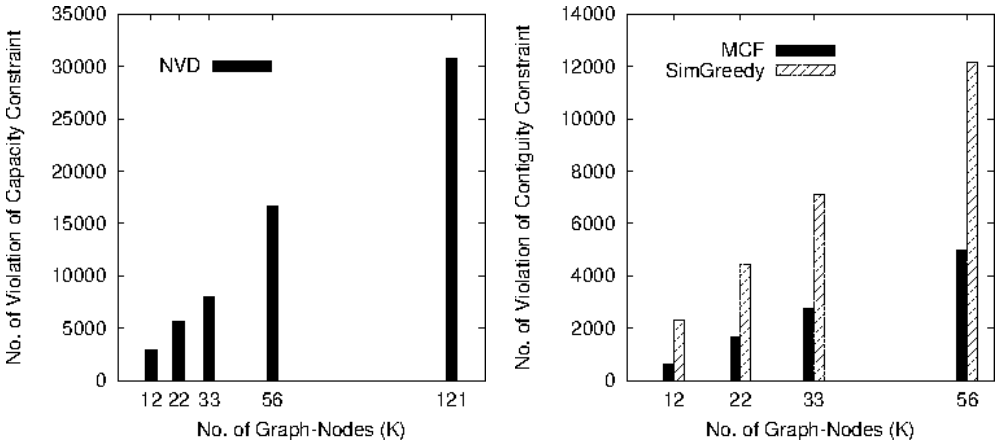
(b) Comparison of Sum of SA shortest Distances

Figure 2.16: Effect of the number of graph-nodes ($|S|= 30$) (MCF and SimGreedy use min-sum and others use SA min-sum)

Constraint Violation under NVD, MCF, and SimGreedy

The third set of experiments examined the rate of constraint violation. First, we fixed the number of service centers to 30 and incrementally increased the number of graph-nodes. Next, we fixed the number of graph-nodes to 32, 744 and varied the number of

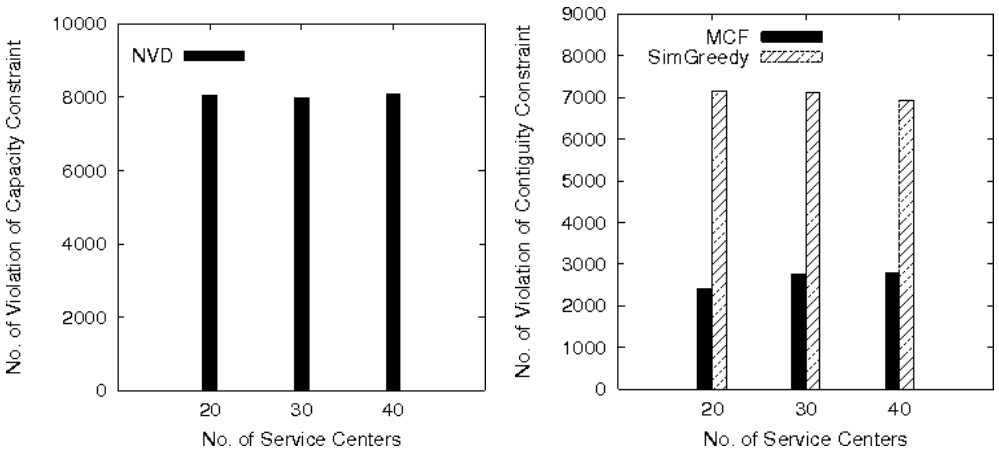
service centers from 20 to 40. Service center locations were chosen randomly and the constraint violations were averaged over 50 test runs. Figure 2.17(a) and 2.17(b) show that the amount of violations increases proportionally with the size of the network. Figure 2.18(a) and 2.18(b) show that the number of service centers does not affect the amount of violations. In our analysis, NVD, MCF, and SimGreedy violated one of the constraints in every test run.



(a) Number of Excess graph-nodes ($|S|= 30$) (b) Number of Disconnected graph-nodes ($|S|= 30$)

Figure 2.17: Capacity Constraint Violation by NVD and Contiguity Constraint Violation by MCF and SimGreedy

We also test how often the PE algorithms fail to create a CCNVD. We used a Twin Cities road map consisting of 56, 168 nodes. We fixed the number of service centers to 30 and randomly chose their locations from the locations of gas stations. The percentage of experimental cases with CCNVD solution is calculated over 300 test runs. Our results show that both PE-BTCC and PE-SSTD create CCNVDs with an average for 86.8 % of cases. PE-Minor creates CCNVDs with an average for 65.4 % of cases. However, NVD, MCF, and SimGreedy create no solution. It was shown that both PE-BTCC and PE-SSTD perform better than PE-Minor in terms of existence of solution.



(a) Number of Excess graph-nodes ($|N|= 32,744$)

(b) Number of Disconnected graph-nodes ($|N|= 32,744$)

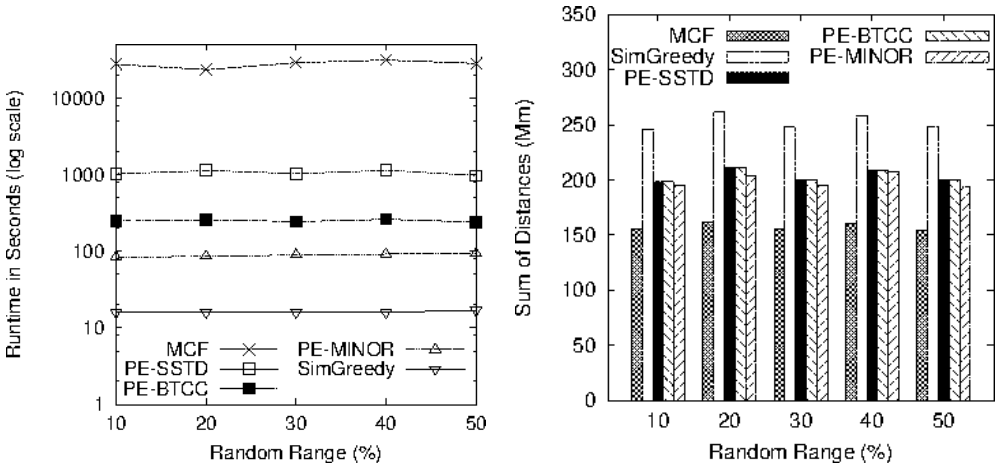
Figure 2.18: Capacity Constraint Violation by NVD and Contiguity Constraint Violation by MCF and SimGreedy

Service Centers of different size capacities

The fourth set of experiments evaluated the performance of the algorithms when service centers had capacities of unequal size. Given a range r % and a capacity c, we randomly chose capacities in the range between $c \times (1 - r)$ and $c \times (1 + r)$. Execution times were averaged over 50 test runs. Figure 2.19(a) shows that unequal sized capacities does not affect performance of the algorithms. Figure 2.19(b) shows that SimGreedy performs poorly with service centers with different size capacities.

Scalability of PE-Minor

The fifth experiment verified the scalability of PE-Minor using continental-sized transportation networks. We chose two regional maps from DIMACS [37]. Table 2.6 shows the number of nodes (e.g., $|N|$), edges (e.g., $|E|$) on the chosen areas, and the number of service centers (e.g., $|S|$) . Service center locations were chosen randomly and execution times were averaged over 50 test runs for each road network. The experiments were performed on an Intel Xeon CPU X5355 machine running CentOS 6.5 with 13 GB of RAM. The PE-Minor algorithm contains the shortest path (SP) computations. Table 2.6 shows the computational time of PE-Minor. It is worth noting that the shortest



(a) Run-time Comparison

(b) Comparison of Sum of SA shortest Distances

Figure 2.19: Effect of Random Capacity ($|N|= 32,744$ and $|S|= 30$) (MCF and SimGreedy use min-sum and others use SA min-sum)

path (SP) computations take the majority time of PE computation time.

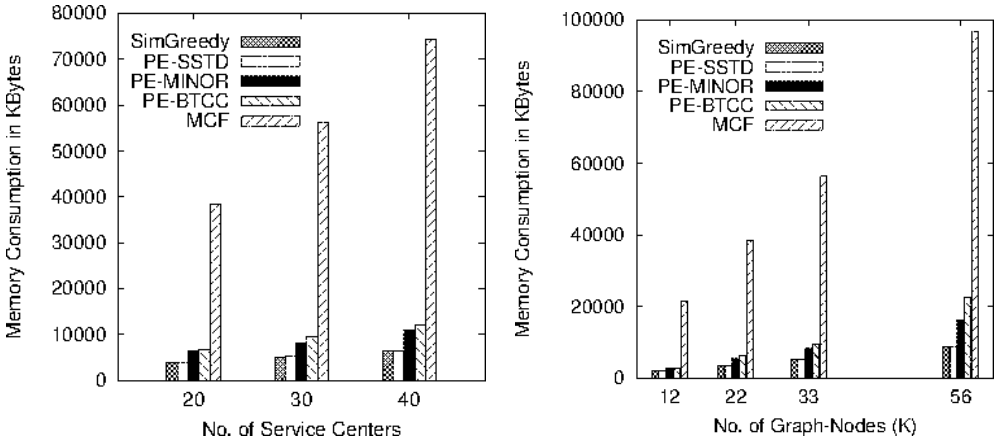
Table 2.6: Scalability with Large Data Sets

Area	$\|N\|$	$\|E\|$	$\|S\|$	Runtime (Min)	
				SP	PE-Minor
Eastern USA	1,262,351	4,085,098	20	40	56
			30	60	76
			40	73	119
Western USA	2,158,871	7,012,158	20	84	122
			30	138	194
			40	167	244

Memory Consumption

The sixth set of experiments measured the memory consumption to run the algorithms. Our analysis used an integer (8byte) data type to store graph elements (e.g., node-id, edge-id, capacity, distance). Performance measurement was the dominant memory consumption during execution. Service center locations were chosen randomly and memory consumption was averaged over 50 test runs. First, we fixed the number of graph-nodes to 32,744 and varied the number of service centers from 20 to 40. Figure 2.20(a) shows

that memory consumption increases for all the algorithms as the number of service centers increases. However, MCF requires by far the most memory. The other algorithms perform much better, with PE-SSTD and SimGreedy consuming slightly less memory than PE-BTCC and PE-Minor. Next we fixed the number of service centers to 30 and varied the number of graph-nodes from 12,402 to 56,168. Figure 2.20(b) shows the effect of the network size. All algorithms consume more memory as the number of graph-nodes increases. However, MCF requires more memory than the other approaches.



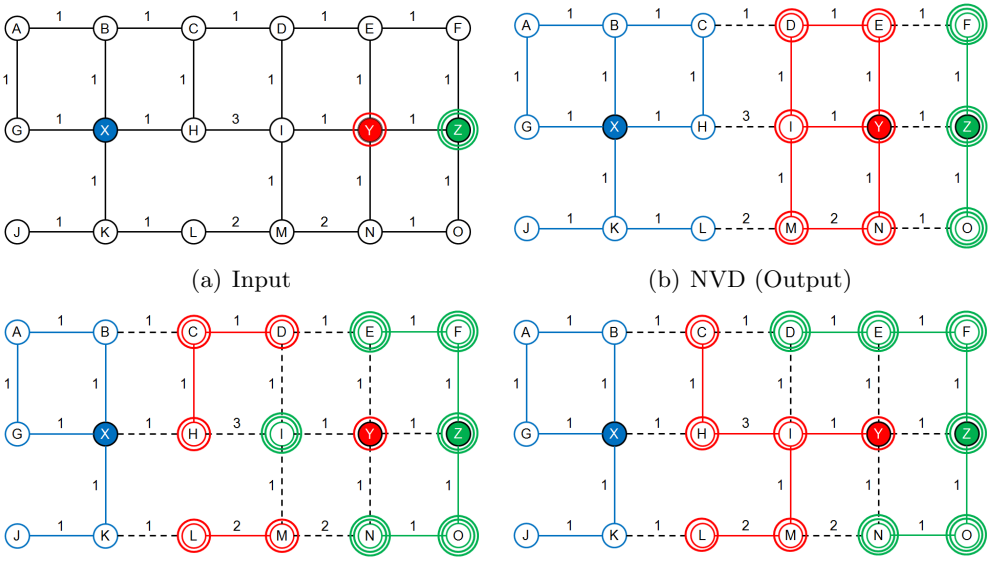
(a) Effect of the number of service centers ($|N|= 32,744$)

(b) Effect of the number of graph-nodes ($|S|= 30$)

Figure 2.20: Memory Consumption Comparison

Discussion

The BTCC achieves a significant computational performance gain over our previous approach. This improvement was obtained using two key data structures: (1) Block Trees and (2) Look-up tables. Given unbalanced allotments (e.g., NVD), our approach smoothly expands (or shrinks) service areas to meet capacity constraints. Using an NVD to represent the contiguous SAs initially allows us to test SA contiguity with the block tree structure. For the SACC problem, we exploit look-up tables to find the articulation nodes in constant time. This novel approach dramatically reduces the computational cost because in each iteration, it is a simple task for the PE algorithm to test the connectivity of the block tree with additional information. We also proposed



(a) Input

(b) NVD (Output)

(c) min-cost flow without SA contiguity (Output) (min-sum=30)

(d) CCNVD (Output) (SA min-sum=32)

Figure 2.21: Example of the Input and Output of NVD, min-cost flow, and CCNVD (Colors show service center allotment)

the PE-Minor to handle continental-sized transportation networks. In our approach, we used a heuristic method (Metis [36]) to create a Graph Minor. The experimental results show that Graph Minor reduces the size of networks and speed-ups the computation of the PE algorithm.

2.5.3 Related Work and Limitations

Previous work on minimizing the sum of the distances between graph-nodes and their allotted service centers can be categorized into two groups: (1) methods that ensure service area (SA) contiguity and (2) methods that honor service center capacity constraints. Prior work on SA contiguity includes the creation of Network Voronoi Diagrams (NVDs), in which each node is assigned to its nearest service center by definition [38, 39, 40]. However, NVDs were not designed to account for capacity constraints. Previous work on honoring service center capacity constraints includes min-cost flow (MCF) approaches [16, 41, 37]. However, these approaches do not always preserve service area contiguity. By contrast, this paper proposes a novel approach for creating

Capacity Constrained Network Voronoi Diagrams (CCNVD) that honors both capacity and contiguity constraints.

Service Area Contiguity	Service center capacity constraints honored: no	Service center capacity constraints honored: yes
no		Min-Cost Flow
yes	Network Voronoi Diagram (nearest center)	Proposed Work

Figure 2.22: Approaches to minimizing the sum of the shortest distances between nodes and their allotted service centers

It is worth noting that the CCNVD and min-cost problems are distinct not only in their constraints but also in their objective function: CCNVD can minimize the sum of the SA shortest distances (SA min-sum), while min-cost flow cannot [41].

Figure 2.21(a) illustrates what node assignment looks like with CCNVD compared to previous approaches. The input is a transportation network (15 graph-nodes ($A, B, \ldots, O$) and three service center nodes (X, Y, and Z)) (Figure 2.21(a)). For simplicity, every graph-node has one unit of demand. Every service center has a capacity to serve 5 graph-node units.

Figure 2.21(b) shows a Network Voronoi Diagram (NVD) allotting every graph-node to the nearest service center. NVD assigns 8 graph-nodes (in blue) to service center X, 5 graph-nodes (in red) to service center Y, and 2 graph-nodes (in green) to service center Z. In this example, NVD does not meet the capacity constraint of service center X, which may lead to delays. Figure 2.21(c) shows the result of a min-cost flow approach. Notice that the service areas for Y and Z are not contiguous. Finally, Figure 2.21(d) shows a Capacity Constrained Network Voronoi Diagram (CCNVD). Both the contiguity and capacity constraints are met in this solution.

2.6 Conclusion and Future work

We presented the problem of creating a Capacity Constrained Network Voronoi Diagram (CCNVD). An important potential application of CCNVD is promoting transportation

resiliency after a disaster. Creating a CCNVD is challenging because of the large size of the transportation network and the constraint that service areas must be contiguous in the graph to simplify communication of service center allotments. In this paper, we describe our Pressure Equalizer (PE) approach for creating a CCNVD that meets the capacity constraints of service centers while maintaining the contiguity of service areas assigned to those centers. Improving PE's scalability to large sized transportation networks requires addressing the computational bottleneck that occurs during Service Area Contiguity Checking (SACC). To remedy the problem, we proposed a novel SACC algorithm, namely Block Tree Contiguity Checking (BTCC), to reduce the computational cost of the PE algorithm. In addition, we proposed a novel Graph Minor approach to handle continental-sized transportation networks. Experiments using five different transportation networks demonstrated that our proposed algorithms significantly reduce the computational cost against our prior work. To simplify the analysis, we assigned a single unit of demand to each node.

In future work, we will study the min-max objective as the PE cost function. Since the min-max minimizes the longest shortest distance from graph-node to their allotted service centers, the PE algorithm should consider new initial assignment and reallotment cost functions. For instance, PE cannot use an NVD as the initial iteration because NVD cannot minimize the min-max. We will also study the capacity (or weight) of the node in terms of the number of units of demand neighboring the node. Since road intersections are used as a proxy measurement for clients, finding a better proxy measurement for client locations is a great idea for the future work. Lastly, we will study a standard integer programming formation of the CCNVD problem.

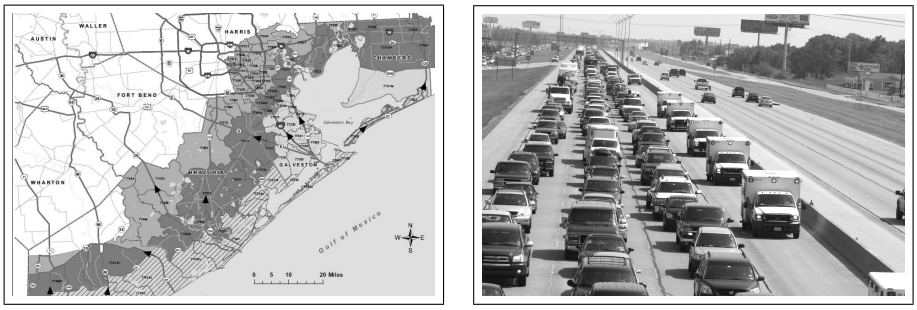
Chapter 3

A Dartboard Network Cut based Approach to Evacuation Route Planning

3.1 Introduction

Hurricane Rita and the recent Tohuku tsunami that hit Japan are reminders that evacuation planning is an essential component of civic emergency preparedness. One of the most important requirements of evacuation planning is to protect population during a disaster. Ensuring the safety of all residents of a structure, city, or region during a disaster requires evacuation planning tools to produce the safest and most efficient route schedules for large scale road networks and populations within limited time constraints. Consider a hurricane evacuation planning problem. Low lying riverside and coastal regions, are especially at risk for a major storm or flooding as shown in Figure 3.1(a). The speed and direction of a hurricane can change rapidly, so the threat to particular areas of the coast may come up suddenly. Massive emergency evacuation from these areas brings more challenges for civic authorities due to the large and unpredictable shape of evacuation zones (EZs) along coastal areas. In 2005, the approach of hurricane Rita provoked one of the largest evacuations in U.S. history, resulting in three million evacuees. During the evacuation, the enormous number of people fleeing from

the Houston area coupled with a number of shortcomings in exit routes for residents caused massive traffic jams. In 1992, Hurricane Andrew, the third most powerful storm to hit the Florida coast caused massive delays and major congestion [42].



(a) Houston Hurricane EZs. Courtesy: http://www.hcoem.org

(b) Congestion From Rita on I-45. Courtesy: FEMA

Figure 3.1: Houston EZ and Congestion from the hurricane Rita

Previously, disasters like Rita and Andrew demonstrated the inadequacy of hand drawn plans for evacuating populations after a disaster. They also demonstrated the need to account for the capacity constraints of road networks. Computational methods of evacuation planning promise more efficient route schedules in the face of massive storms. These methods must be scalable and able to produce results easily in a short time frame. Furthermore, they must be able to handle dynamic environments.

Over the last two decades there has been a considerable amount of research on route planning for evacuation zones and other event scenarios. Recent work on evacuation route planning can be divided into three categories: (1) Linear Programming (LP) methods that use a network flow problem to minimize the total evacuation time [43, 44, 45, 46], (2) Simulation methods that model the evacuation route as individual movements [47, 46] or a traffic assignment problem [48], and (3) Heuristic methods that use an approximate optimization technique to minimize the computation time. The LP approach uses iterative algorithms (e.g., simplex or ellipsoid method) to minimize the cost function based on given constraints [43, 44, 45]. This approach requires using a static network model for a dynamic environment to generate optimal evacuation plans. Consequently, the transportation network needs to be transformed into a time-expanded graph (TEG) by constructing $T + 1$ copies of nodes and edges [49]. Unfortunately, the

number of variables and iterations in this linear program is in general exponential in the size of the underlying network, limiting its usefulness for large scale networks [50]. Simulation methods use individual traveler behaviors or traffic assignment in greater detail including the interaction between vehicles. One problem with this model is that regulating individual movements or assigning traffic flow associated with Wardrop's equilibrium model [51] in emergency evacuation is very complicated, making it inappropriate for large evacuation scenarios. Finally, heuristic methods can be used to incorporate capacity constraints into route planning and find near-optimal evacuation plans with reduced computational cost. This is useful for medium-size transportation networks within a limited amount of time. A well known approach for this category is the Capacity Constrained Route Planner (CCRP) [52, 10]. However, CCRP incurs excessive computational cost for large network and population datasets.

Recent approaches may not be able to scale up to large size transportation networks on densely populated regions due to the limited capacity constraints of road networks and large numbers of evacuees. New insights are needed to reduce the high computational costs where these methods incur with large-scale networks. Our work focuses on minimizing computation time and enhancing scalability for large transportation networks. We explore a novel routing algorithm that exploits the underlying spatial network structure of road networks. A common evacuation scenario displays dartboard network structure leading to be partitioned by dartboard network cuts (DBN-cuts). We introduce the notion of dartboard network structure and explain how to organize and group evacuation routes. Our new approach accelerates the routing algorithm by grouping multiple node-independent shortest routes to reduce the number of search iterations. For example, instead of a single-route shortest-path algorithm, we use a node-independent shortest-paths algorithm to aggregate evacuees on different spatial locations and evaluate evacuation routes without sacrificing the quality of the evacuation route plan.

Our contributions: In this paper, we propose a novel algorithm that uses an underlying dartboard network structure driven by DBN-cuts. DBN-cuts group multiple node-independent shortest routes and reduce iterations of an evacuation routing algorithm. We use a generalized node-independent shortest path algorithm to obtain these node-independent routes and minimize the computational time. Specifically, our

contributions are as follows:

- We propose a dartboard network structure based on common evacuation scenarios.
- We propose a dartboard network-cut for evacuation route planning (DBNC-ERP) algorithm to group multiple node-independent routes based on DBN-cuts.
- We provide a cost model for the DBNC-ERP.
- We experimentally evaluate the proposed algorithm and validate the cost model using real road network datasets.

Scope and outline: This paper proposes a novel evacuation route planning algorithm for large scale network datasets based on dartboard network structure. Our approach uses node-independent shortest routes for DBNC-ERP. The rest of the paper is organized as follows: Section 3.2 provides the problem definition. Section 3.3 presents our proposed approach. In Section 3.4, we discuss how our cost model to predict the performance could be derived. Section 3.5 describes the experiment design and presents the experimental observations and results. Finally, Section 3.6 concludes the paper.

3.2 Problem definition

The problem of evacuation route planning can be formalized as follows: Given a transportation network with maximum node and edge capacity constraints, initial node occupancy, and destination locations, our objective is to find evacuation route scheduling that minimizes the evacuation time and minimizes the computational cost. We formally define the problem as follows:

Input: A transportation network with

- non-negative integer capacity constraints on nodes N and edges E,
- the total number of evacuees and their initial location, and
- location of evacuation destination

Output: An evacuation plan consisting of a set of origin-destination routes and a scheduling of evacuees on each route.

Objective:

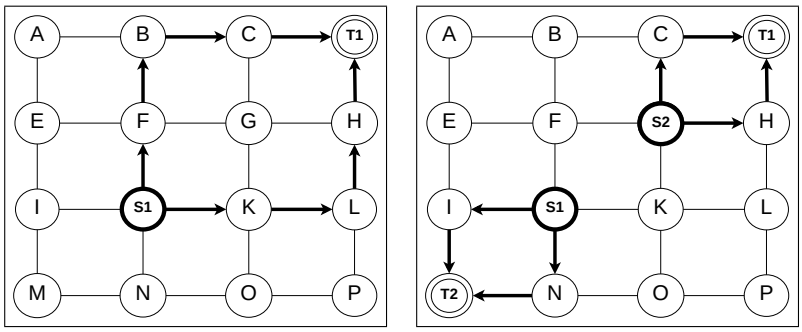
- Minimize the computational cost of producing the evacuation plan

- Minimize the evacuation time.

Constraints:

- Edge travel time preserves FIFO (First-In First-Out) property.
- The scheduling of evacuees on each route observes the capacity constraints.
- Limited amount of computer memory

3.3 Dartboard Network Cuts for Evacuation Route Planning



(a) One source and one destination (b) Two sources and two destinations

Figure 3.2: Node-independent routes in a grid-like network

Basic Concept: Spatial networks are represented and analyzed as a graph composed of nodes N and edges E. Every node N represents spatial location in geographic space with a number of evacuees and a node capacity. Every edge E represents a connection between two nodes and has a travel time with an edge capacity. A sequence of nodes $n_1, n_2, n_3 \ldots, n_n$ is called a path (or route) if there is an edge between two consecutive nodes. A tree is an undirected graph in which any two nodes are connected by exactly one simple path. A forest is a disjoint union of trees. A set of paths from a source node s to a destination node d is node-independent path if none of the paths share any nodes aside from s and d. In Figure 3.2(a), for example, there are two node-independent paths traversing from $S1$ to $T1$ ($S1 \rightarrow F \rightarrow B \rightarrow C \rightarrow T1, S1 \rightarrow K \rightarrow L \rightarrow H \rightarrow T1$). Figure 3.2(b) shows four node-independent paths based on two pairs of source and

destination nodes ($S1 \rightarrow I \rightarrow T2, S1 \rightarrow N \rightarrow T2, S2 \rightarrow C \rightarrow T1, S2 \rightarrow H \rightarrow T1$). These node-independent paths are not necessarily unique so that there may be more than one way of choosing a set of independent paths.

Theorem 3. *Given a pair of nodes u, v, the upper bound of the number of node-independent paths is min(the degree of a node u, the degree of a node v).*

Proof. Let m be $degree(u)$ and n be $degree(v)$. First, assume that $m \geq n$ and there exist n independent paths. When we add one more independent path, no incoming edge of n exists to obtain the independent path. Second, assume that $m < n$ and there exists m independent paths. Again, we cannot add one more independent path because there is no available outgoing edge of m. Consequently, the maximum of node-independent paths is bounded by $min(degree(u), degree(v))$. □

Why are node-independent routes important for capacity constrained route planning algorithms ? The key idea is that node-independent routes never share each other's capacity constraints at the same time during the route evaluation process.

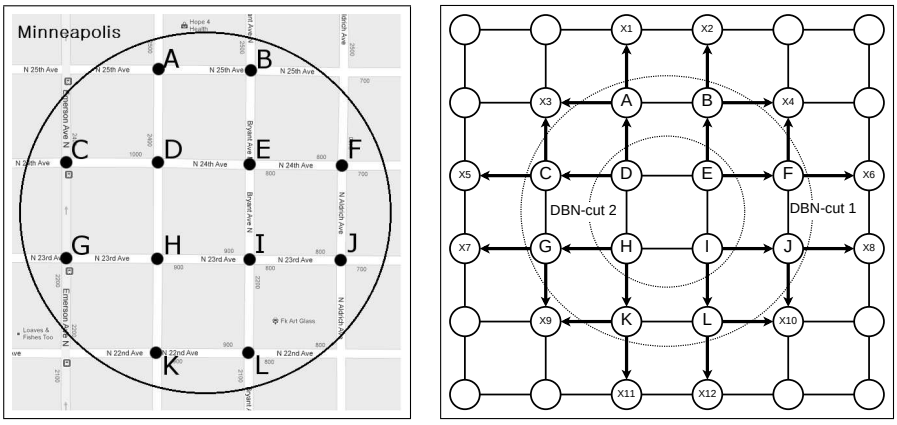
3.3.1 Dartboard Network Structure

Spatial road networks were shaped in response to socioeconomic activities maximizing ease of navigation in the areas. The structures are neither trees nor perfect grids, but a combination of these structures that emerges from the social and constructive processes. The networks may be broken down into independent routes: most simply, routes that do not share any local parameters, such as node and edge capacity. These independent routes can partition evacuees and use discreet flows to compute the evacuation routes. Consider, for example, the grid-like road network in Figure 3.2. Because independent routes do not share the capacity constraints of other roads in the network, one node-independent shortest path algorithm for these independent routes can minimize the computational time. Unfortunately, many road networks have low degree intersections, making it hard to retrieve many node-independent routes. It is known that the mean degree of intersections in the US interstate road network is only about 2.86 [23]. By Theorem 3, we can retrieve at most 2.86 node-independent routes. One way to remedy the low degree issue is to use super nodes for source nodes and destination nodes. For

instance, in Figure 3.2(b), $S1$ and $S2$ are grouped into a super source node S, and $T1$ and $T2$ are grouped into a super destination node T. Consequently, one node-independent shortest path algorithm for S and T can compute four node-independent shortest routes as increasing node degree of S and T.

The spatial network organization of a place has an extremely important effect on the way people move through space and time. Evacuation route planning involving large numbers of evacuees has a well defined evacuation zone (EZ) (e.g., entire cites or coastal plains), making it possible to find the spatial movement patterns on transportation networks. For example, in Figure 3.3(a), the circular EZ encloses 12 nodes and all the travelers on these nodes need to move out of the circle. The network has one unit of capacity with one unit of travel time and two evacuees per node. Figure 3.3(b) shows the network model for the EZ; The nodes inside of the EZ are $(A, B, C, D, E, F, G, H, I, J, K, L)$ and the nodes outside of the EZ are $(X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12)$. The nodes in the EZ are divided into two groups by a DBN-cut. The outside nodes (A, B, C, F, G, J, K, L) have more shorter available routes compared to the inside nodes (D, E, H, I), reflecting an "outer first, inner last" flow pattern. That means, after evacuees from boundary areas move out of outer boundary areas, there is a secondary wave of evacuees from inner regions into boundary areas that will prepare to move out of the EZ. To characterize this pattern, dartboard network structure is defined as a network organization partitioned according to Dartboard Network Cuts (DBN-cuts), shown in Figure 3.3(b). To explain how this structure happens, look at the arrows in Figure 3.3(b). Given the EZ and number of evacuees, the evacuation plan needs to maximize the number of evacuees using the available shortest routes shown as arrows at each time step $t \in T$. As can be seen in Figure 3.3(b), the outer group (A, B, C, F, G, J, K, L) is poised to flee first from the EZ to maximize the number of evacuees, followed by the inner group (D, E, H, I), which takes its place and is similarly set to flee to maximize the number of evacuees again. We define a dartboard network cut (DBN-cut) as a cut in the flow of travelers in an evacuation network such that all the travelers in a single group are removed at the same time.

Theorem 4. *In a dart board network structure with FIFO property and a limited capacity constraint, a maximal dynamic flow algorithm for evacuation planning moves the outside nodes first and goes to inside nodes incrementally.*



(a) EZ in Minneapolis. Courtesy: http://maps.google.com

(b) Network Model for EZ

Figure 3.3: Dartboard network structure in road networks

Proof. Assume that evacuees from inside nodes arrive at the destination before evacuees at outside nodes. This means that some outside nodes must have had to wait to exit the boundary area in order to make sure there is available capacity for evacuation of inside nodes. Otherwise, there would not have been enough capacity available for the inside nodes to exit the EZ through the outer boundary area. This implies an increase in the total time required to evacuate all people, thereby violating our objective to minimize the evacuation time. □

Table 3.1: Evacuation route plan based on dartboard network structure in Figure 3.3(b)

Group	Source	# of Evacuees	Route Node Id(Time)	Arrival Time	Group	Source	# of Evacuee	Route Node Id(Time	Arrival Time
1	A	1	A(0)-X1(1)	1	1	K	1	K(0)-X9(1)	1
		1	A(0)-X3(1)				1	K(0)-X10(1)	
	B	1	B(0)-X2(1)			L	1	L(0)-X10(1)	
		1	B(0)-X4(1)				1	L(0)-X12(1)	
	C	1	C(0)-X3(1)		2	D	1	D(0)-A(1)-X1(2)	2
		1	C(0)-X5(1)				1	D(0)-C(1)-X5(2)	
	F	1	F(0)-X4(1)			E	1	E(0)-B(1)-X2(2)	
		1	F(0)-X6(1)				1	E(0)-F(1)-X6(2)	
	G	1	G(0)-X7(1)			H	1	H(0)-G(1)-X7(2)	
		1	G(0)-X9(1)				1	H(0)-K(1)-X11(2)	
	J	1	J(0)-X8(1)			I	1	I(0)-J(1)-X8(2)	
		1	J(0)-X10(1)				1	I(0)-L(1)-N12(2)	

Table 3.1 shows the results of an evacuation route plan with dart-board network structure. Each row in the table describes the schedule of a group of evacuees moving together to arrive at destinations at time step $t \in T$. During each iteration, the algorithm tries to group the node-independent routes and maximize the number of evacuees. For example, group 1 aggregates 16 node-independent shortest routes for 32 evacuees and group 2 aggregates 8 node-independent shortest routes for 16 evacuees. Note that each group aggregates evacuation routes on different spatial locations and reaches destinations at the same time.

Given the number of routes and the evacuation time, our principal objective is to maximize the number of evacuees for each time step $t \in T$. To achieve this end, an evacuation routing algorithm attempts to maximize the available shortest routes at each time step $t \in T$. Given this basic assumption, in each time step t, many node-independent routes may exist to maximize the number of available shortest routes. In the next subsection, we introduce our new algorithms to efficiently create DBN-cuts on evacuation networks.

3.3.2 DBNC-ERP algorithm

In this subsection, we describe our node-independent shortest paths approach for a dartboard network structure. A naive approach is to enumerate all available routes and remove node-dependent routes. However, the search space becomes exponential for combinations of the available multiple routes. General approaches for node-independent shortest paths construct a shortest path tree (SPT) and check the node dependency for each route [53, 54, 55]. The SPT of order n nodes has size $n - 1$, resulting in reduced search space by examining the boundary nodes [56]. In our problem, there are many source nodes to traverse in order to reach destination nodes. Instead of a SPT, we consider a shortest *forest* where each tree has a different root to handle multiple source nodes and iteratively choose node-independent shortest routes having available capacity. A forest of order n with k roots (or source nodes) has size $n-k$ since not every forest shares nodes. We allow SPTs in the forest to share the same destinations because evacuees from different sources may reach the same destination. For each route, possible waiting time at each node is considered due to limited capacity constraint. Algorithm 3 shows a way to identify the evacuation routes with node-independent shortest paths.

The input for the pseudo code is an evacuation transportation network consisting of nodes, edges, source nodes, sink nodes, and capacity constraints. The output is an evacuation route schedule containing a sequence of nodes and edges. All source nodes and destination nodes are grouped by two super nodes to increase the node degree (Line 2). A shortest forest is constructed to find node-independent routes (Line 3,4). After retrieving the node-independent routes, the capacity constraints are applied to these routes and available routes for evacuees are chosen (Line 5,6). The next step is to reduce node and edge capacities along the routes (Line 7,8) and repeat the above process until we finish finding the routes for all remaining evacuees.

The DBNC-ERP algorithm based on node-independent routes may need several iterations before obtaining available routes at each time step $t \in T$. Our greedy approach is related to an aspect of DBN-cuts that attempts to maximize the evacuation routes for each time step t. Line 4 in Algorithm 3 evaluates all available routes based on the "share-nothing" property of node-independent routes and reduces iterations for constructing the shortest forest.

Algorithm 3 Pseudo code for DBNC-ERP

Inputs: - A set of nodes N and edges E with capacity constraints C
- Each edge $e \in E$ has a travel times t.
- A set of source nodes S including initial evacuee occupancy O and a set of destination nodes D

Outputs: Evacuation plan including route schedules of evacuees on each route r

DBNC-ERP Algorithm:

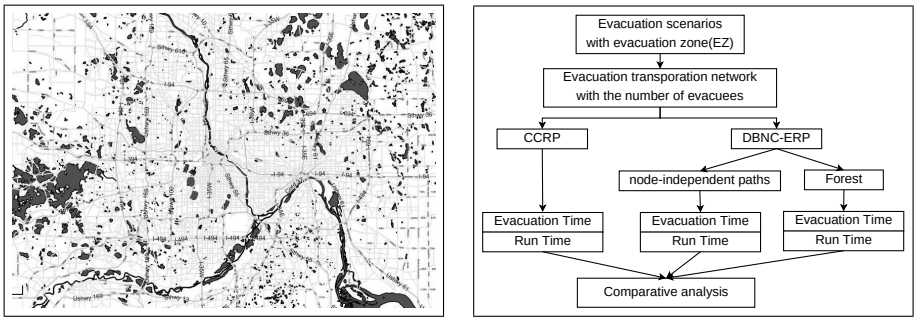
1: **while** any source node $s \in S$ has evacuees **do**
2: group all source nodes with a super source node ss and group all sink nodes with a super sink node sd.
3: construct a shortest forest from S to D based on ss and sd. Every tree can share the destination D.
4: find all routes R that are shortest node-independent paths from S to D.
5: **for** $r \in R$ **do**
6: compute the minimum route capacity C_{min} with the edge and node capacity c along the route r.
7: evacuee flow $f = min$(number of remaining evacuee at s in r , C_{min}).
8: reduce the node and edge capacity c along the route r using f.
9: remove evacuees from O using f.
10: **end for**
11: **end while**

3.4 Algebraic Cost Model of DBNC-ERP

The goal of this section is to present cost models for DBNC-ERP for estimating computational cost based on node-independent routes. The transportation road network can be modeled as a grid-like network that has many alternative shortest routes [57, 58]. In general, there are at least two node-independent routes between any pair of nodes [59]. In our analysis, we use 2 as a lower bound of road network connectivity. Assume that n is the number of nodes, m is the number of edges, and p is the number of evacuees. The DBNC-ERP iteratively chooses k node-independent shortest routes and reserves the capacity for these routes. In the worst case, one evacuee can traverse the route, resulting in p/k iterations. DBNC-ERP constructs a shortest forest using a modified Dijkstra's algorithm and super nodes. The worst case computational time for a dense graph is $O(n^2)$. For sparse networks, Dijkstra's algorithm can be implemented in time $O(n \log n)$ [43, 44]. Basically, the node-independent shortest routes are computed as the same bounds for Dijkstra's algorithm [60, 61]. In our approach, we put super nodes to group the source nodes and destination nodes, then construct a shortest forest instead of a SPT. Capacity constraint checking and updating takes $O(kn)$ for k node-independent shortest routes. The cost model of the DBNC-ERP algorithm is $O((p/k) \cdot n \log n)$. In transportation road networks, we can compute the lower bound of DBNC-ERP as $O((p/2) \cdot n \log n)$.

The cost model shown above is the strictly lower bound. This is because this model does not consider the DBN-cuts which group source nodes in different spatial locations, leading to increase degree of a source node. If the network has sufficiently large numbers of destination nodes, then the number of node-independent routes is bounded by the degree of a super source node, according to Theorem 3. This point is easily illustrated by a circular evacuation zone in Figure 3.3. From the boundary of the EZ to its center, DBNC-ERP incrementally groups source nodes using DBN-cuts and attempts to find the available evacuation routes for each source of the group. This reduces iterations of DBNC-ERP by the number of DBN-cut groups. However, the number of groups for DBN-cuts is highly dependent upon the underlying network structure of the EZ.

3.5 Experimental Evaluation



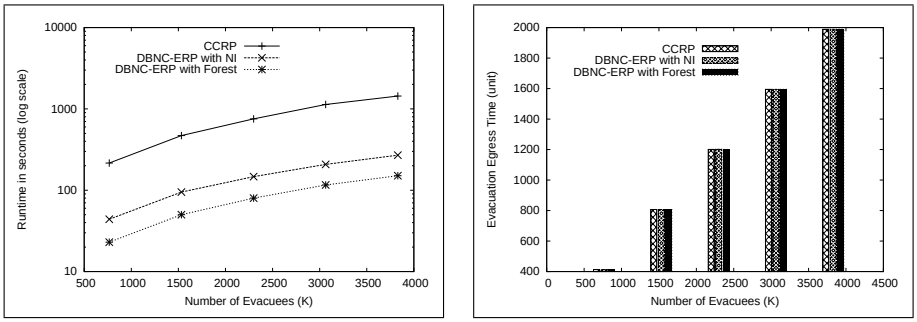
(a) Minnesota Road Network (b) Experiment layout

Figure 3.4: Experiment setup for evacuation routing planning

Figure 3.4(b) shows our experimental setup. Our experiments used a Minneapolis, MN road map consisting of $8,868$ nodes and $24,126$ edges, taken from TIGER/Line [62]. The software was implemented in Java 1.7 with 1 GB memory run-time environment. All experiments were performed on an Intel Core i7-2670QM CPU machine running MS Windows 7 with 8 GB of RAM. We used two evacuation zones (EZs): one for a circular area and the other for a riverside area. Given the location of an incident and its scope R, we defined a circular EZ as the circular area centered at the incident with radius R and a riverside EZ (or coastal EZ) as the buffer area with R adjacent to rivers. We tested three different approaches. The first two are CCRP [10] and DBNC-ERP with node-independent shortest paths (DBNC-ERP with NI). The third approach was DBNC-ERP with a shortest forest, as a candidate for DBNC-ERP to attempt to maximize the number of evacuees at each time step $t \in T$. The property of node-independent routes is easily exploited to reduce the computational time by reducing each iteration. However, the DBNC-ERP algorithm with node-independent shortest paths uses two route scans to evaluate the availability: one scan for node dependency and the other for capacity constraints. Intuitively, the forest for node-independent routes displays partially node-independent. We may relax the node-independent constraints and remove the node-dependency check for the forest. We call this method DBNC-ERP with a shortest forest. The strength of this approach is that it reduces route checking time when the number of available routes is large. The main disadvantage is that it

may yield false-positive node-independent routes, which will be removed during capacity checking time.

3.5.1 Experimental Observation and Results



(a) Runtime comparison

(b) Evacuation time for each evacuee number

Figure 3.5: Effect of the number of evacuees

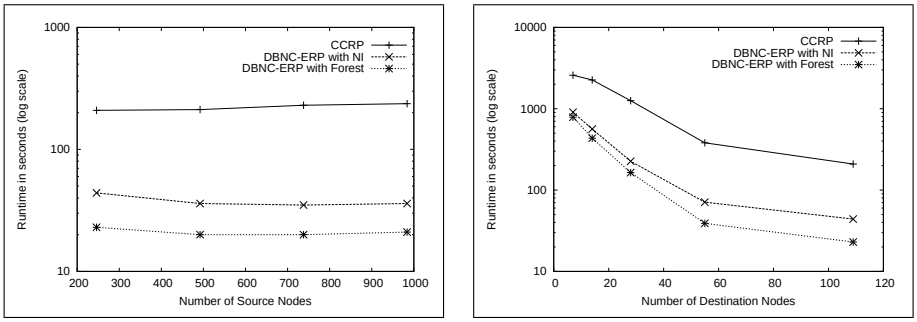
Experiment 1: Effect of the number of evacuees

The purpose of the first experiment was to evaluate the effect of number of evacuees on the performance of the algorithms. We fixed the number of source and destination nodes and multiplied the evacuees of each node. The experiment was done using networks of 246 source nodes, 109 destination nodes, and $1,847$ nodes for a circular EZ. We incrementally increased the number of evacuees from $766,123$ to $3,064,492$. Figure 3.5(a) shows that the two DBNC-ERP approaches outperform CCRP. As increase of number of evacuees, the performance gap also increases. This is because the DBNC-ERP approaches group the node-independent routes to minimize the iterations. DBNC-ERP with a forest shows slightly better performance compared to DBNC-ERP with IN due to the longer node-dependency checking time. Figure 3.5(b) shows that all three algorithms were not distinguished in terms of evacuation time results. As the number of evacuees grows, the egress time increases.

Experiment 2: Effect of the number of source nodes

The second experiment evaluated the effect of the number of source nodes on the performance of the algorithms. We fixed the number of destination nodes and the number of evacuees. To increase the number of source nodes, source nodes were made to share

the evacuees to new source nodes. The experiment was done using networks of 109 destination nodes, $1,847$ nodes for a circular EZ, and $766,123$ evacuees. We incrementally increased the number of source nodes from 246 to 984. Figure 3.6(a) shows that number of source nodes has little effect on algorithm performance. Nevertheless, the two DBNC-ERP approaches run faster than CCRP.



(a) Effect of the number of source nodes (b) Effect of the number of destination nodes

Figure 3.6: Effect of the number of source and destination nodes

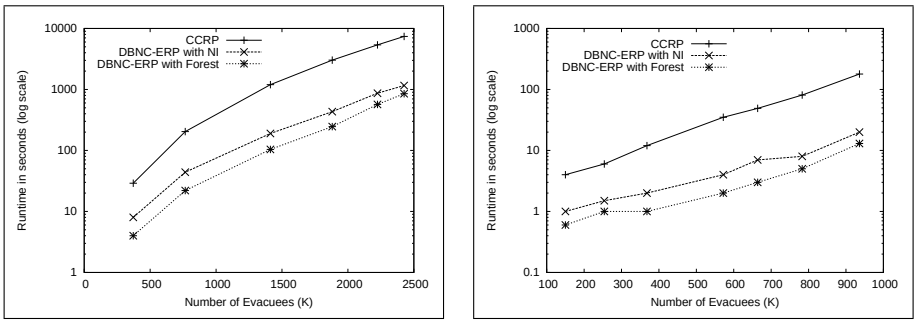
Experiment 3: Effect of the number of destination nodes

The third experiment evaluated the effect of the number of destination nodes on the performance of the algorithms. We fixed the number of source nodes, and the number of evacuees and decreased the number of destination nodes. The experiment was done using networks of 246 source nodes, $1,847$ nodes for circular EZ, and $766,123$ evacuees. Figure 3.6(b) shows that as the number of destination nodes grows, the runtime decreases. As the number of destination nodes increases, the performance gap also increases according to Theorem 3.

Experiment 4: Scalability for large network datasets

The fourth experiment evaluated scalability for large network datasets. We incrementally increased the radius of the circular EZ from $5km$ to $30km$. Figure 3.7(a) shows that the runtime of DBNC-ERP scaled well to these large network sizes. These results show that runtime can be reduced by up to 80%

Experiment 5: Effect of shape of EZ



(a) Circular EZs (b) Riverside EZs

Figure 3.7: Scalability on different EZ shapes

The fifth experiment evaluated the effect of the shape of the EZ on the performance of the algorithms. If we put the destination nodes as boundary nodes of the EZ, a circular EZ will have a least possible number of boundary nodes due to its small surface area. If the EZ shows irregular shape (e.g., coastal areas), the number of boundary nodes increases as the surface area of the EZ increases. In our experiment, we chose evacuation zones along rivers as shown in Figure 3.1(a) and incrementally increased the length of the non-circular EZ to cover the entire length of MN rivers. Results showed that the two DBNC-ERP approaches ran faster on the riverside EZ (Figure 3.7(b)) than on the circular EZ (Figure 3.7(a)). This is because the number of boundary nodes of the irregular shaped EZ is greater than of the circular EZ. Our results show that the runtime can be reduced by up to 90% when our approach is applied to irregularly shaped evacuation zones.

Experiment 6: Effect of spatial network structure of EZs

The last experiment explored the effect of different spatial structures of EZs. We chose five different coastal or isolated regions taken from OpenStreetMap [35] and assigned a synthetic population for each EZ. Table 3.2 shows that the maximum speed up in DBNC-ERP algorithms is bounded by the number of dartboard network cuts enclosing the EZ (i.e., the number of destination nodes). Once again, the fewer the destination nodes (e.g., Key West and Galveston) for EZs, the fewer the node independent routes to speed up.

Table 3.2: Experimental Result for other regions

Region	Runtime reduction	# of nodes	# of edges	# of dart board network cuts	# of evacuees
Key West,	64%	1,291	3,809	3	25,820
Galveston, TX	82%	4,146	12,368	3	36,675
Jackson, WY	85%	673	1,696	13	9,422
Cape Cod, MA	92%	32,257	80,438	30	225,799
San Francisco, CA	96%	16,409	48,058	78	810,084

3.5.2 Discussion

The proposed DBNC-ERP algorithm advances the state of the art computational techniques for evacuation route planning. The proposed algorithm achieves a significant computational performance gain over current techniques. This improvement was obtained using three key features of the underlying road networks: (1) FIFO with limited capacity, (2) "outer-first" flow pattern, and (3) dartboard network structure.

The first, two features implicitly assume that the risk in a given EZ is distributed uniformly. The DBNC-ERP algorithm uses an incremental strategy for such EZs, where people in the outer region of the EZ are evacuated first. This leads to reductions in overall evacuation time. However, an incremental strategy may not be suitable for EZ's with non-uniform risk (e.g. point based threats such as bombs). Typically, in such scenarios, the evacuation proceeds in phases, where evacuees on the inner ring of nodes are evacuated first. This type of protocol may violate the "outer-first" and FIFO assumption. We plan to explore such multi-phase evacuation scenarios in the future.

The third feature, dartboard network structure, allows the DBNC-ERP algorithm to reserve capacities along k paths per iteration. Here, k is the number of node-independent routes. During the execution of the algorithm, parameter k manifests itself as the number of dartboard network cuts in the given EZ. Our results to date provide evidence for a correlation between the size of dartboard network cuts and performance gain. For example, the proposed algorithm achieved a reduction of 92% in runtime for Cape Cod which had a cut set of size 30. On the other hand, the Key West dataset with a cutset size of 3, allowed a reduction of 64%.

Due to time limitations, we have only used five geographic areas for preliminary evaluation of the proposed algorithm. In the future, we plan to test our algorithm on a larger number of geographic areas to characterize this correlation between the spatial structure (dartboard network cuts) of the road network and the computational running time.

3.6 Conclusion and Future work

Evacuation route planning for large transportation networks is becoming increasingly important for dealing with man-made and natural disasters, such as hurricanes, terrorist acts, and nuclear accidents. An important component of evacuation planning methods is the ability to account for capacity constraints of the road network with manageable computational cost. In this paper, we introduced dartboard network structure to reflect evacuee flow pattern for common evacuation scenarios by exploiting the spatial structure of the road network. Based on dartboard network structure, our DBNC-ERP algorithm partitions the network using dartboard network cuts (DBN-cuts) and groups source nodes in different spatial locations to maximize the number of evacuees. We also showed the cost model to explain how to reduce the computational cost based on dartboard network structure. Experimental evaluation of DBNC-ERP demonstrated significant improvements over previous work.

Chapter 4

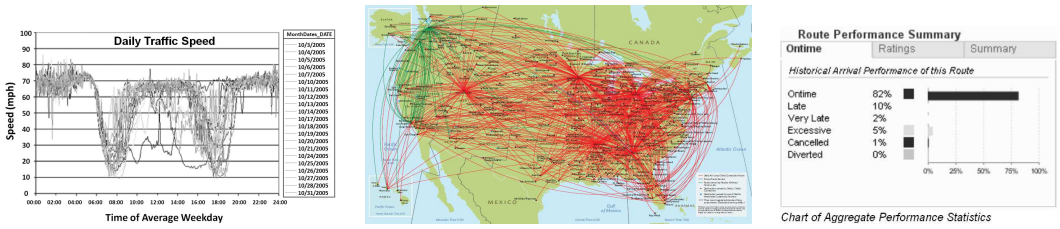
Lagrangian Approaches to Storage of Spatio-temporal Network Datasets

4.1 Introduction

Spatio-Temporal Network (STN) can be defined as a graph $G = (N, E, T)$, where N is a set of nodes, E is a set of edges connecting two nodes, and every node and edge is associated with temporal information T (e.g., left-turn restriction and travel cost). STN datasets are becoming indispensable in societal applications, such as surface and air transportation management systems. One of the most appealing properties of these datasets is their ability to capture network attributes that vary over time. Consequently, STN datasets are usually massive in size and are accessed based on spatio-temporal movement patterns, making I/O efficient storage and access methods a significant challenge.

4.1.1 Representative Application Domains

As an example, let us consider surface and air transportation management systems. The Federal Highway Administration [6] is recording traffic data of major roads and highways using sensors, such as loop detectors, across the United States. Depending



(a) Traffic speed measurements over 30 days on a portion of highways. Courtesy:www.fhwa.dot.gov

(b) US travel routes for Delta Airlines. Courtesy: www.airlineroutemaps.com

(c) Flight Statistics. Courtesy: www.flightstats.com

Figure 4.1: Examples of Spatio-Temporal Networks and Data

on the type of sensor, traffic levels are recorded as often as every minute or less, as shown in Figure 4.1(a). The Mobility Monitoring Program (MMP), started in 2000 by the Texas Transportation Institute, evaluated the use of sensors for traffic information around the United States. By 2003, the MMP was receiving traffic sensor data from over 30 cities and $3,000$ miles of highway, with sensor readings occurring roughly every 30 seconds. These data are recorded 24 hours a day, 365 days a year, resulting in millions of time steps per year for each sensor. In 2004, MMP published a report citing the need for better processing and storage of historical traffic data in order to benefit traffic management [6].

Airlines connect thousands of destinations across the world through various routes between airports. Maintaining accurate records of route performance is essential to evaluating and ensuring timely airline service, along with analyzing the potential causes and effects of delays. Figure 4.1(b) shows flight routes between major US airports. In order to measure route characteristics, such as average delay, each flight along the route is recorded with parameters such as flight time, departure time, landing time, etc. This flight information creates a STN that allows historical queries to be answered, such as the one shown in Figure 4.1(c) which describes how often a flight is 'on time', 'late', 'very late', or 'excessively late'. Other more complex queries, such as how a delay on a particular route affects connecting flights, can also be analyzed with this data.

4.1.2 Problem Definition

The problem of Storing Spatio-Temporal Networks (SSTN) can be formalized as follows: given a spatio-temporal network (STN) and a set of STN operations, our objective is to find a storage scheme that minimizes the I/O costs of the operations. We formally define the problem as follows:

Input:

- A spatio-temporal network S
- A set of spatio-temporal operations O (e.g., $LGetAllSuccessors()$)

Output:

- Data Partitioning of S, across data pages

Objective:

- Minimize data page access for operations in O

Constraints:

- S is too large for storage in main memory.
- Temporal-edge attribute information needs to be preserved.

In this paper, we focus on a special case of the SSTN problem, namely SSTN-G∀S, that optimizes the $LGetAllSuccessors()$ operation for retrieving all successors for a given node on a STN. The SSTN-G∀S problem is NP-hard and a proof is provided in Section 4.3.1. Intuitively, this problem is computationally challenging because of the fixed data page size, the large size of STN datasets, and the constraint that data page access for the STN operation must be minimized.

In our prior work [63], we addressed a different special case, namely SSTN-G1S, that optimizes the $LGetOneSuccessor()$ operation for retrieving a single successor for a given node on a STN. In that work, we provided a non-orthogonal partition method (LCP-G1S) as a solution of the SSTN-G1S problem.

4.1.3 Related Work and Limitations

Over the last decade, considerable work on STNs has focused on pre-computation techniques and speed-up algorithms for time-dependent route planning [64, 65, 66, 67, 68, 69]. By comparison, there has been relatively little work on the design and evaluation of storage and access methods for STNs. Early work employed geometric space indexes for both space and space-time [70]. Orthogonal partitioning methods, such as the longitudinal or snapshot method [69], are able to capture network connectivity based on either space or time orthogonally as shown in Figure 4.2. The longitudinal method stores temporally consecutive properties of a node (or edge) into the same data page whereas the snapshot method stores a topologically connected sub-graph for a given time instance into the same data page. Current related work for storing and accessing STN data have relied on these orthogonal approaches [69].

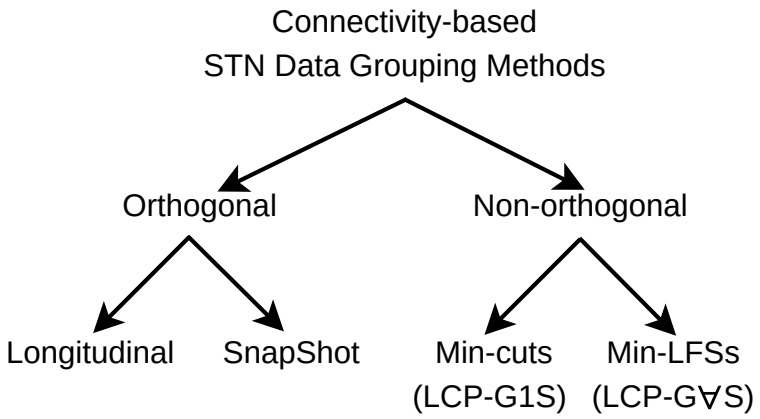


Figure 4.2: Connectivity based STN Data Grouping Methods

In contrast with these methods, our approach focuses on non-orthogonal partitioning based on movement-connectivity. This method stores neither common node (or edge) nor common time instance information into the same data page. Our preliminary work proposed a storage and access method for retrieving a single successor node from a parent node (LCP-G1S) and significantly reduced I/O costs compared to orthogonal approaches [63]. This paper proposes a complementary storage and access method for efficiently retrieving all successor nodes at one time. In our new method, we define the concept of a Lagrangian Family Set (LFS) that groups a parent node and all successor

nodes together.

4.1.4 Our Contributions

In this paper, we propose a solution to the SSTN-G∀S problem. Our contributions are as follows:

- We prove that the SSTN-G∀S problem is NP-hard.
- We propose a novel storage and access method, namely LCP-G∀S, using the concept of a Lagrangian Family Set (LFS).
- We analyze the algebraic cost of the LCP-G∀S algorithm.
- We experimentally evaluate the proposed approach using real-world and synthetic STN datasets.

4.1.5 Scope and Outline

We propose a new method for storing STNs into a data file with efficient data partitioning for STN operators. Secondary indexing techniques are not considered, as they can be applied on top of the data file. This paper focuses on optimization of the $LGetAllSuccessors()$ operation on STNs. For simplicity of discussion, it does not consider work related to time dependent shortest-path algorithms or indexing data structure for STNs. Industry is examining and implementing approaches for storing STNs based on orthogonal partitioning and sharing of time-series between edges and other lossy compression techniques. These 'speed profiles' are not examined in this paper, as we focus on full data storage as it is recorded from sensors. In addition, the intent of our work is not to evaluate industry choices but to explore conceptual ideas relevant to storage of STNs. If the size of a record is greater than the size of data page, the record can be made to span more than one data page by adding a data page pointer at the end of the record.

The rest of this paper is organized as follows: Section 4.2 describes basic concepts of spatio-temporal networks and non-orthogonal partitioning. Section 4.3 describe the LCP-G∀S approach using the concept of a Lagrangian Family Set. In Section 4.4, we

provide algebraic analysis of our proposed approach. Section 4.5 details our experimental evaluation on real-world and synthetic datasets. Finally, we conclude and discuss future work in Section 4.6.

4.2 Basic Concepts

4.2.1 Spatio-temporal Networks and Lagrangian Paths

A spatio-temporal network (STN) can be represented as a spatial graph with temporal attributes, composed of nodes (N), edges (E), and discrete time steps (t). Every node N represents a spatial location in geographic space. Every edge E represents a connection between two nodes and has temporal attributes associated with it. A sequence of nodes $n_1, n_2 \ldots, n_n$ is called a path (or route) if there is an edge between every two consecutive nodes.

STN queries can be described with a Lagrangian frame of reference [71]. We define a Lagrangian Path as a spatio-temporal network path where each edge is associated with time-varying attributes. For example, a time-varying attribute such as travel time $d(e,t)$ can be associated with an edge $e(n_1, n_2)$ and interpreted as follows: if t is the departure time from node n_1, then $d(e,t)+t$ is the arrival time at node n_2. This implies that travel time on a Lagrangian path changes as time progresses. In Figure 4.3, for example, at $t = 1$, edge AC has a travel time of 2. At $t = 2$, the edge value AC decreases to 1, indicating a reduced travel time for that section of the route. Consider traveling a route $A \rightarrow C \rightarrow D$. When starting at $t = 1$, it takes 3 time steps to reach D. However, when starting at $t = 2$, the same trip requires only 2 time steps.

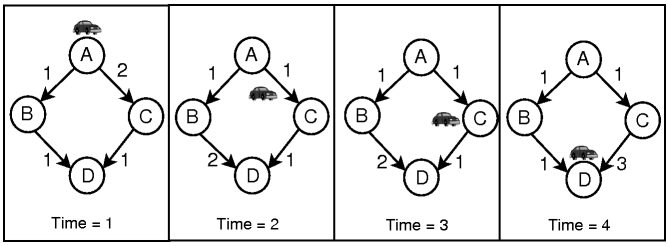


Figure 4.3: Snapshot model of a spatio-temporal road network

A STN dataset can be modeled as a time-expanded graph (TEG), that is, a spatio-temporal model which replicates each node along the time series such that a time-varying attribute is represented between replicated nodes [72, 73]. In a TEG, every node is associated with a temporal attribute, and every edge joining two nodes represents the spatio-temporal relation between the two nodes. Figure 4.4 illustrates a TEG for the spatio-temporal road network shown in Figure 4.3. In Figure 4.4, each edge contains the travel time needed to traverse the road segment. Given a start time of $t = 1$, a route $A \rightarrow C \rightarrow D$ consists of edge AC at $t = 1$ and edge CD at $t = 3$. The derived graph forms a directed, acyclic graph and neatly expresses a data access sequence for Lagrangian-connectivity.

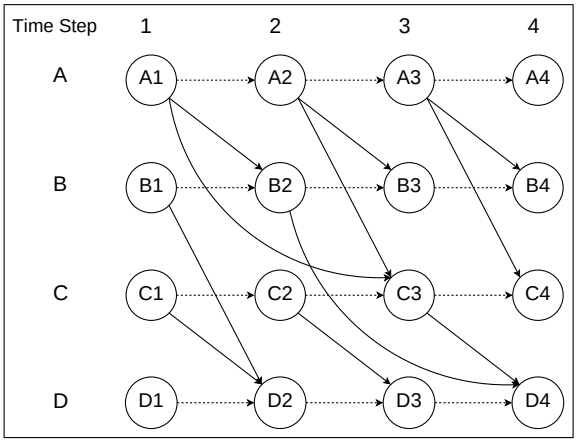


Figure 4.4: STN as a time expanded graph

Table 4.1: Access Operations for Spatio-Temporal Networks

	"At" time (Single time instance)	**"During" time (Multiple time instances)**
Single Successor	$LGetOneSuccessor(Node(n,t), n_s)$	$LGetOneSuccessor(Node(n,t1), Node(n,t2), n_s)$
All Successors	$LGetAllSuccessors(Node(n,t))$	$LGetAllSuccessors(Node(n,t1), Node(n,t2))$

4.2.2 Sub-node Data Structure of STN Datasets

Network representation is a crucial component of network analysis such as path computation and route evaluation. In our study, we use an adjacent-list oriented representation

to support connectivity-based computation algorithms. For a STN, the node record format $Node(n,t)$ consists of a node-id n, a time step t, and time-varying information (e.g., turn restrictions and allowable waiting time) as well as outgoing edges for its successors (sub-nodes). Each edge has time-varying information such as travel time $d(e,t)$. We refer to this data structure as a sub-node data structure.

4.2.3 Spatio-Temporal Network Operations

STN operations retrieve spatio-temporal topology relationships between each node record stored in the database. The basic operations used in our approach to analyze STN datasets are: $FindNode()$, $LGetNodeTransition()$, $LGetOneSuccessor()$, and $LGetAllSuccessors()$.

FindNode(n,t): Given a node-id n and a time step t, the operation searches for the data page pointer (or block-id) associated with the record from the secondary index and retrieves the data page containing the desired node record $Node(n,t)$ from the buffer cache or disk. If the data page is located in the buffer cache, the buffered data page is used. If it is not, the data page is fetched from the disk and stored in the buffer cache. After that, the node record $Node(n,t)$ is extracted within the data page.

LGetNodeTransition($Node(n,t)$): Given a node record $Node(n,t)$, this operation retrieves the same node at the next time step, $Node(n,t+1)$. The same-node transitions can be implemented by calling a $FindNode(n,t+1)$. Alternatively, if the node record $Node(n,t)$ contains a data page pointer for the transition node, the operation goes directly to the data page and retrieves the node.

LGetOneSuccessor($Node(n,t), n_s$): Given a node record $Node(n,t)$ and a single successor-id n_s, this operation finds the successor's time step (t_s) from the node record $Node(n,t)$. Then it calls $FindNode(n_s,t_s)$ to retrieve $Node(n_s,t_s)$. Alternatively, if node record $Node(n,t)$ contains a data page pointer for successor n_s, the operation goes directly to the data page and retrieves $Node(n_s,t_s)$ without calling $FindNode(n_s,t_s)$. When the desired record is located in the buffer cache, no additional disk I/O is needed.

LGetAllSuccessors($Node(n,t)$): Given a node record $Node(n,t)$, it finds all successor-ids (n_s) and their time steps (t_s) from the node record $Node(n,t)$. Then it calls $FindNode(n_s,t_s)$ for every successor. Alternatively, if the node record $Node(n,t)$ contains data page pointers for incident records, the operation goes directly to these data pages and retrieves all successors without calling multiple $FindNode(n_s,t_s)$. If the node

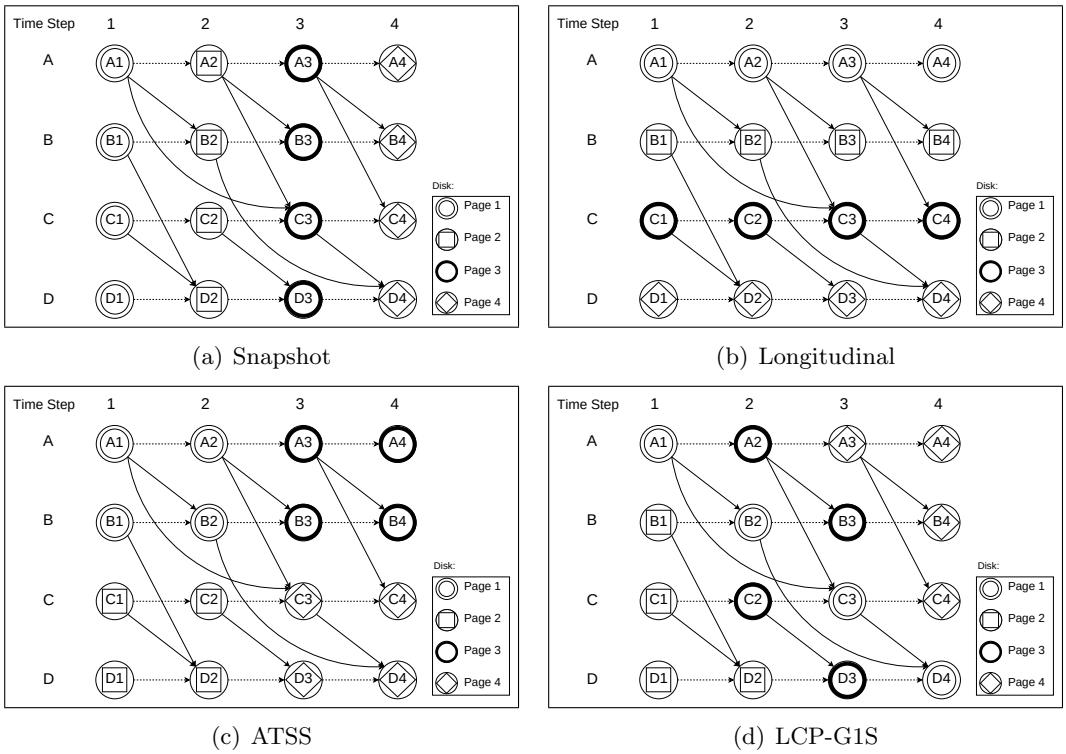
records of successors can be made to share the same data page as $Node(n, t)$, we can significantly reduce the additional I/O cost of $LGetAllSuccessors()$.

In our access method, the $LGetAllSuccessors()$ operation is optimized by exploiting the buffer cache. First, all data page pointers (or block-ids) for successors are searched from the secondary index or the node record, and node records of successors located in the buffer cache are retrieved first. After that, remaining record-pointers are sorted by data page pointer, and the relevant data pages are sequentially fetched from disk. This procedure can reduce the need for additional I/O even though there is only one buffer cache.

Table 4.1 lists common STN operations. These operations can be viewed as functions that map time steps onto topological locations on a STN space. In Figure 4.4, for example, calling $LGetOneSuccessor(A1, B)$ returns one successor $B2$. $LGetOneSuccessor(A1, A3, B)$, on the other hand, returns $B2$, $B3$, and $B4$ for the interval $[A1, A3]$. $LGetAllSuccessors(A1)$ retrieves $B2$ and $C3$, whereas $LGetAllSuccessors(A1, A3)$ examines the interval $[A1, A3]$ and returns $B2$, $B3$, $B4$, $C3$, and $C4$. Our work focuses on efficient storage and access methods for the two STN operations ($LGetAllSuccessors()$ and $LGetOneSuccessor()$). In the following sub-section, we review a detailed description of our proposed approaches.

4.2.4 Non-Orthogonal Partitioning of STNs

As the length of a time series in a STN increases, efficient execution of traversal queries requires a different approach for storing the data on disk. Traditional approaches partition networks based on nodes using some orthogonal emphasis (e.g., temporal or spatial). However, such methods do not work well with large STNs. Consider, for example, evaluating a route ($A \rightarrow C \rightarrow D$) at a time step of 1 on the STN, shown in Figure 4.4. With the orthogonal partitioning methods, such as snapshot (Figure 4.5(a)) and longitudinal (Figure 4.5(b)), whenever an edge is traversed ($A1 \rightarrow C3$ and $C3 \rightarrow D4$), a disk I/O is needed to retrieve the data page containing the record for the next node. One naïve candidate to handle this issue is the aggregated time-stamped snapshot (ATSS) method that partitions the STN with static network connectivity and divides the time-series information into temporal chunks (Figure 4.5(c)). The ATSS method can be seen as a trade-off between snapshot and longitudinal partitioning. Our previous work [63]



(a) Snapshot

(b) Longitudinal

(c) ATSS

(d) LCP-G1S

Figure 4.5: STN Partitioning Methods

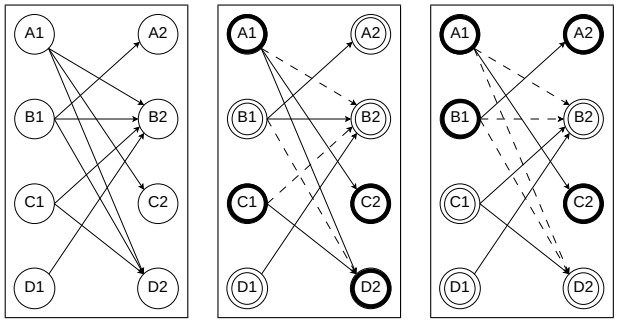
showed that ATSS was practical when the travel time was fairly uniform. However, the main disadvantage is that it is not possible to determine the appropriate time interval parameter value to yield a better performance.

Figure 4.5(d) shows the LCP-G1S as a solution of the SSTN-G1S problem. It is a non-orthogonal partitioning that optimizes for retrieving a single successor for a given node on a STN. By performing a min-cut graph partitioning [74], we can create partitions based on single edge connectivity on a STN. As a result, spatio-temporally connected nodes can be collocated together on data pages. We showed that LCP-G1S results in more efficient I/O when calling a *LGetOneSuccessor*() operation from Table 4.1 or queries composed of them [63]. In this example, with the LCP-G1S method, traversing from node $A1$ to $C3$ and then $C3$ to $D4$ requires only one data page as all relevant sub-node records are collocated on the same data page.

4.3 LCP-G∀S: Lagrangian-Connectivity Partitioning for LGetAllSuccessors()

In this section, we introduce our new method to optimize the $LGetAllSuccessors()$ operation and explain key elements to design the algorithm in detail. We first point out that a solution of the SSTN-G1S problem is not entirely appropriate for the $LGetAllSuccessors()$ operation due to the structure of the set of successors.

Consider the STN example in Figure 4.6(a). There are four parent nodes (e.g., $A1, B1, C1, D1$) and four successor nodes ($e.g., A2, B2, C2, D2$). We define a blocking factor as the number of node records per data page. In this example, we use a blocking factor of 4. We also assume that the data page for a parent node is located in the buffer cache before calling a STN operation (e.g., $LGetAllSuccessors()$ or $LGetOneSuccessor()$).



(a) Input (STN) (b) Output (SSTN-G1S) (c) Output (SSTN-G∀S)

Node	I/O Cost: LGetAllSuccessors() SSTN-G1S	I/O Cost: LGetAllSuccessors() SSTN-G∀S
A1	1	1
B1	1	1
C1	1	0
D1	0	0
Total	3	2
Avg.	0.75	0.5

Page 1
Page 2
Edge Cut

Edge		I/O Cost: LGetOneSuccessor() SSTN-G1S	I/O Cost: LGetOneSuccessor() SSTN-G∀S
A1	B2	1	1
A1	C2	0	0
A1	D2	0	1
B1	A2	0	0
B1	B2	0	1
B1	D2	1	1
C1	B2	1	0
C1	D2	0	0
D1	B2	0	0
Total		3	4
Avg.		0.333	0.444

(d) I/O Costs of STN operations

Figure 4.6: Comparison of SSTN-G∀S and SSTN-G1S

Figure 4.6(b) shows a solution of the SSTN-G1S problem and Figure 4.6(c) shows a solution of the SSTN-G∀S problem. The dashed line represents the edge-cut that connects two nodes stored in different data pages. The additional disk I/Os for a STN operation is measured by the number of data page fetches for the operation call. Consider the $LGetAllSuccessors(A1)$ operation in Figure 4.6(b). $A1$ needs one more

data page fetch to access $B2$, thereby causing one additional disk I/O for retrieving the set of successors.

The result of SSTN-G1S has three edge-cuts (Figure 4.6(b)) whereas SSTN-G∀S has four edge-cuts (Figure 4.6(c)). By contrast, Figure 4.6(d) shows that SSTN-G1S requires 0.75 additional disk I/Os on average for the *LGetAllSuccessors*() operation, whereas SSTN-G∀S requires 0.5 additional disk I/Os. Note that although the edge-cuts in SSTN-G∀S are greater than SSTN-G1S, SSTN-G∀S shows better performance than SSTN-G1S in terms of the I/O costs for the *LGetAllSuccessors*() operation. This is true because the SSTN-G1S ignores the grouping between a parent and its successor set. For this reason, we propose a Lagrangian-Connectivity Partitioning method for *LGetAllSuccessors*() (LCP-G∀S). The basic idea of LCP-G∀S is to minimize the number of distinct data pages for a parent and its successors set. We refer to the set of data pages for a parent and its successors as a *Lagrangian Family Set (LFS)*.

4.3.1 SSTN-G∀S Objective Function and Problem Hardness

In this subsection, we define the SSTN-G∀S objective function for minimizing the cost of *LGetAllSuccessors*(). Then we prove its NP-hardness.

Proposition 1. *The expected I/O cost of a LGetAllSuccessors() operation is minimized by minimizing* $\sum_{n \in N} |LFS(n)|$*, where* $LFS(n)$ *is a data page-id set for a node* n *and its successors.*

Proof. Let $p(n)$ denote the page-id of the node n. After calling $Find(n)$, the data page storing the node n is transferred into the buffer cache. Clearly, the successors that share the same data page as the node n do not require additional disk I/Os. On the other hand, the successors stored in $LFS(n) \backslash \{p(n)\}$ cause additional $|LFS(n)|-1$ disk I/Os. This implies the disk I/Os for *LGetAllSuccessors*() is minimized by minimizing $\sum_{n \in N} |LFS(n)|$. □

Consider the example again in Figure 4.6. The *LFS* for all parent nodes by SSTN-G1S requires, on average, 1.75 data pages (Figure 4.6(b)), while our new problem needs only 1.5 (Figure 4.6(c)). It demonstrates the advantages of our proposed SSTN-G∀S objective function.

The NP hardness of SSTN-G∀S follows from a well-known result about the NP-hardness of the following balanced hypergraph k-partitioning problem [75, 76].

Given a graph $G = (N, E)$, where N denotes a set of nodes and E a set of hyperedges that can connect more than two nodes, the goal of balanced hypergraph k-partitioning problem is to partition N into equal sized parts $N_1, \ldots, N_k$ while minimizing hyperedge-cuts. A hyperedge is not cut if all nodes are in one partition, and cut exactly once otherwise. This graph partitioning problem is already NP-complete for the case $k = 2$, which is also called the balanced hypergraph bi-partitioning problem [75, 76].

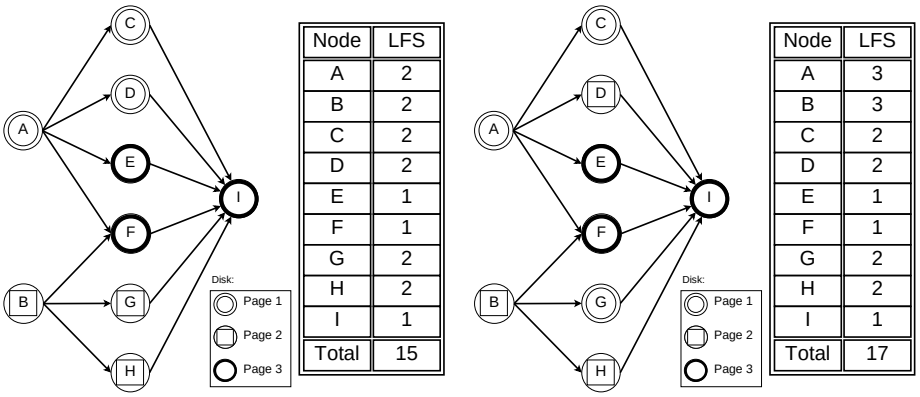
Theorem 5. *The SSTN-G∀S problem is NP-hard.*

Proof. Given $STN(N, E, T)$, the SSTN-G∀S problem partitions N into equal sized parts $N_1, \ldots, N_k$ while minimizing $\sum_{n \in N} |LFS(n)|$. The SSTN-G∀S problem clearly belongs to NP since given an instance of SSTN-G∀S and the maximum bound B, we can take a set of parts such that $\sum_{n \in N} |LFS(n)|$ is lower than B as a valid certification. Consider the case $k = 2$. Then it is easy to show that SSTN-G∀S problem is equivalent to the balanced hypergraph bi-partitioning problem because $LFS(n)$ becomes a hyperedge that connects node n and its successors. Since the balanced hypergraph bi-partitioning problem is constructed from SSTN-G∀S ($k = 2$) in polynomial time, the proof is complete. □

Generally speaking, the SSTN-G∀S problem is more difficult than the hypergraph partitioning problem. Consider the example in Figure 4.7 ($k = 3$). The result of SSTN-G∀S shows that the sum of $LFSs$ is 15 (Figure 4.7(a)), whereas hypergraph partitioning may yield 17 (Figure 4.7(b)). As expected, SSTN-G∀S should minimize data page fragment inside LFS as well as minimize hyperedge-cuts. In the next section, we the define basic concept underlying the LCP-G∀S algorithm.

4.3.2 Basic Concept for LCP-G∀S algorithm

Our objective function can be generalized for frequent network operations in terms of expected I/O cost. Consider the case that the weight $w(n, t)$ associated with $Node(n, t)$ represents the relative frequency of a query accessing $Node(n, t)$. Then, every $LFS(n)$ has a weight corresponding to the access frequencies of a node n. We now generalize and formally describe our objective function that underlies the frequency of query



(a) SSTN-G∀S (hyperedge-cuts=6) (b) Hypergraph Partitioning (hyperedge-cuts=6)

Figure 4.7: Hardness of SSTN-G∀S

occurrence.

$$SSTN{-}G\forall S(w(n)) = minimize \sum_{n \in N} w(n)|LFS(n)|, \tag{4.1}$$

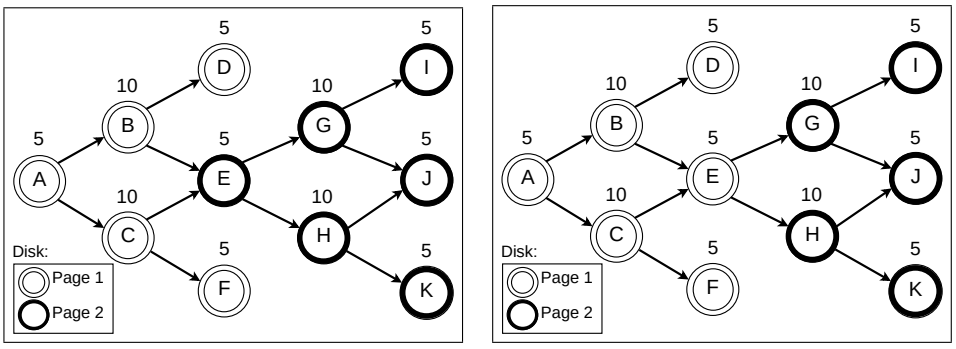
where $w(n)$ represents the relative frequency of the $LGetAllSuccessors()$ operation that accesses the node n.

Our objective for SSTN-G∀S is to minimize $\sum_{n \in N} w(n)|LFS(n)|$ while preserving page-size constraints (e.g., maximum page size and minimum page utilization ratio) on each partition. One of the best-known partitioning algorithms is a Tabu-search based iterative-improvement algorithm (e.g., Kernighan-Lin(KL) and Fiduccia-Mattheyses (FM)) [77, 78]. The algorithm begins with an initial partition and iteratively moves a node n to improve the objective function. Using the idea of a FM algorithm [78], the gain (or cost) of moving a node n_1 to another partition is defined as the change in $\sum_{n \in N} w(n)|LFS(n)|$ before and after moving the node n_1. Then, the gain of moving the node n_1 from page A to page B can be represented by

$$Gain_{A \to B}(n_1) = \sum_{n \in STN_{P(n_1)=A}} w(n)|LFS(n)| - \sum_{n \in STN_{P(n_1)=B}} w(n)|LFS(n)|, \tag{4.2}$$

where $STN_{P(n_1)=A}$ is the $STN(N, E)$ with the page-id of $n_1 = A$.

Example: Figure 4.8 shows a STN and $Gain_{2 \to 1}(E)$. The nodes on the left side are



(a) Before moving node E (b) After moving node E

Figure 4.8: Move node E from Page 2 to Page 1 ($Gain_{2\to1}(E) = 15$)

stored in Page 1 and the rest are stored in Page 2. The numbers above the nodes represent the relative frequency of a query accessing the nodes. After calling $FindNode(n)$, the additional disk I/Os for $LGetAllSuccessors(n)$ operations for all nodes in Figure 4.8(b) are 15 less than in Figure 4.8(a). The result of moving the node E is exactly the same as the result of the gain function ($Gain_{2\to1}(E) = 95 - 80 = 15$).

The difficulty with the above gain function in practice is in computing all $|LFS(n \in STN)|$. We now show that $LFS(n_1)$ and $LFS(n \in pred(n_1))$ are sufficient to compute the gain for the moving node n_1.

Proposition 2. *The gain of moving the node n_1 from page A to page B is defined as:*

$$Gain_{A\to B}(n_1) = \sum_{\substack{n\in\{n_1\}\cup \\ pred(n_1)}} w(n)(|LFS(n, STN_{P(n_1)=A})| - |LFS(n, STN_{P(n_1)=B})|), \tag{4.3}$$

where $LFS(n, STN_{P(n_1)=A})$ is $LFS(n)$ with $STN_{P(n_1)=A}$ and $pred(n)$ is a predecessor set of a node n.

Proof. If $LFS(n)$ contains a node n_1, then the node n should be n_1 or an element of $pred(n_1)$. Let $X(n_1) = \{x | x \in N, x \neq n_1, x \notin pred(n_1)\}$. Clearly, $\forall x \in X(n_1)$ have no change of their gain after moving the node n_1 because $LFS(x)$ and $LFS(n \in pred(x))$ do not contain the node n_1. Therefore, we do not need to consider $LFS(n \in X(n_1))$ for the gain of the node n_1. □

The moving node n_1 will change other node gains. However, examining all nodes is

not an efficient way is to find these nodes. It is only worth examining candidate nodes that may change their $LFSs$.

Proposition 3. *The movement of node n_1 changes the gain of its predecessors, its siblings, its successors, and itself.*

Proof. According to Proposition 2, the movement of node n_1 changes only $LFS(n_1)$ and $LFS(n \in pred(n_1))$. Therefore, we need to consider other node gain updates that use $LFS(n_1)$ or $LFS(n \in pred(n_1))$. Clearly, node n_1 uses both $LFS(n_1)$ and $LFS(n \in pred(n_1))$, predecessors of node n_1 use $LFS(n \in pred(n_1))$, siblings of node n_1 use $LFS(n \in pred(n_1))$, and successors of node n_1 use $LFS(n_1)$. □

We refer to predecessors, siblings, and successors of a node n as *LCP-G∀S Neighbors of a node n* (*LCP-G∀SNbrs(n)*). After moving a node n_1, we simply update the gain of the entire node $n \in LCP{-}G\forall SNbrs(n_1)$.

Finally, we define the notion of a boundary between partitions, which helps to reduce unnecessary examinations of nodes.

Proposition 4. *If $|LFS(n)|= 1$ and $|LFS(m \in pred(n))| = 1$, then the gain of any movement of node n is negative.*

Proof. $LFS(n \in N) = 1$ implies that all nodes in $LFS(n)$ are stored in the same data page. After moving node n, the decrease of the gain is $-(w(n)+ \sum_{m \in pred(n)} w(m)) < 0$. □

We can divide all nodes into two parts: LCP-G∀S inside nodes ($LCP{-}G\forall SINs(N)$) and LCP-G∀S boundary nodes ($LCP{-}G\forall SBNs(N)$).

- $LCP{-}G\forall SINs(N) = \{n | n \in N,\ |LFS(n)| = 1 \text{ and } |LFS(m \in pred(n))| = 1\}$
- $LCP{-}G\forall SBNs(N) = \{n | n \in N,\ |LFS(n)| > 1 \text{ or } |LFS(m \in pred(n))| > 1\}$

LCP-G∀S boundary nodes may have positive gain or not, but LCP-G∀S inside nodes never have a positive gain. In our approach. we consider a set of LCP-G∀S boundary nodes to optimize our objective function.

Example: As can be seen in Figure 4.8(a), all nodes except node B, C, and E are LCP-G∀S inside nodes. Intuitively, no nodes $n \in LCP{-}G\forall SINs(N)$ cause additional disk I/Os for *LGetAllSuccessors*(). After moving node E to Page 1, E, G, and H become LCP-G∀S boundary nodes, as shown in Figure 4.8(b).

4.3.3 LCP-G∀S algorithm

In this subsection, we describe our LCP-G∀S algorithm using Tabu-search based iterative optimization. A Tabu-search explores candidate solutions and iteratively chooses the best one from a Tabu-list. It allows temporary deterioration in solution quality to escape from local optima, but eventually achieves a near-optimal solution [79, 80]. It is important to note that a high-quality initial solution reduces the overall run time of the algorithm [80]. In our algorithm, we choose the result of LCP-G1S (i.e., min-cut graph partitioning) as an initial solution.

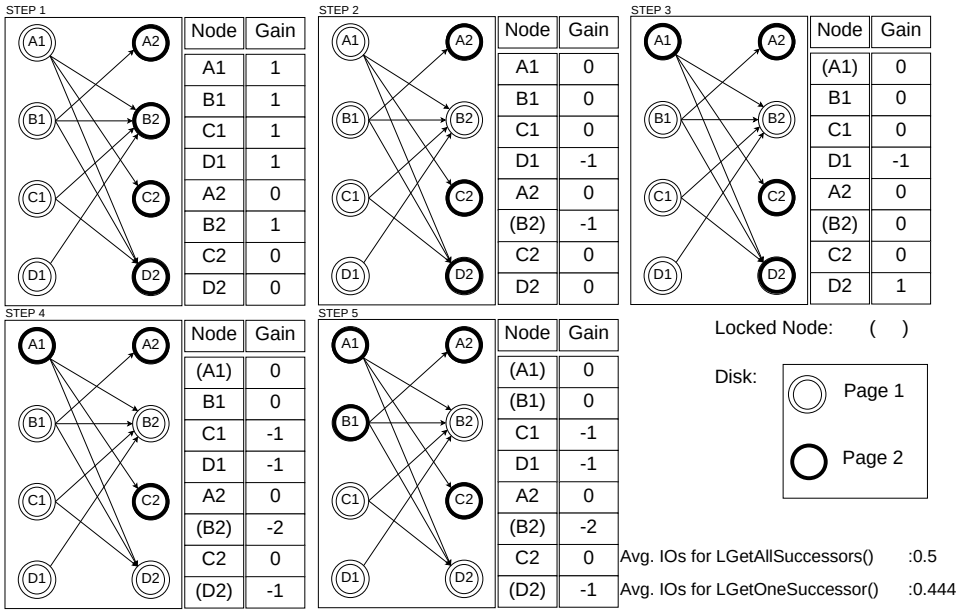


Figure 4.9: Two-way LCP-G∀S

An example of one pass of a two-way LCP-G∀S algorithm is shown in Figure 4.9. To simplify the example, we use orthogonal partitioning as an initial solution and enforce a node balancing constraint (i.e., every partition has the same number of nodes). In every step, LCP-G∀S computes gains for LCP-G∀S boundary nodes and chooses the best movement based on both the largest gain and the balancing constraint. After a node is moved, it is locked to prevent moving in the remainder of the pass. In the example, $A1$, $B1$, $C1$, $D1$, and $B2$ have the largest gain (e.g., gain = 1) at Step 1. As a tie break rule, we assume that a successor has a higher priority to move to the other

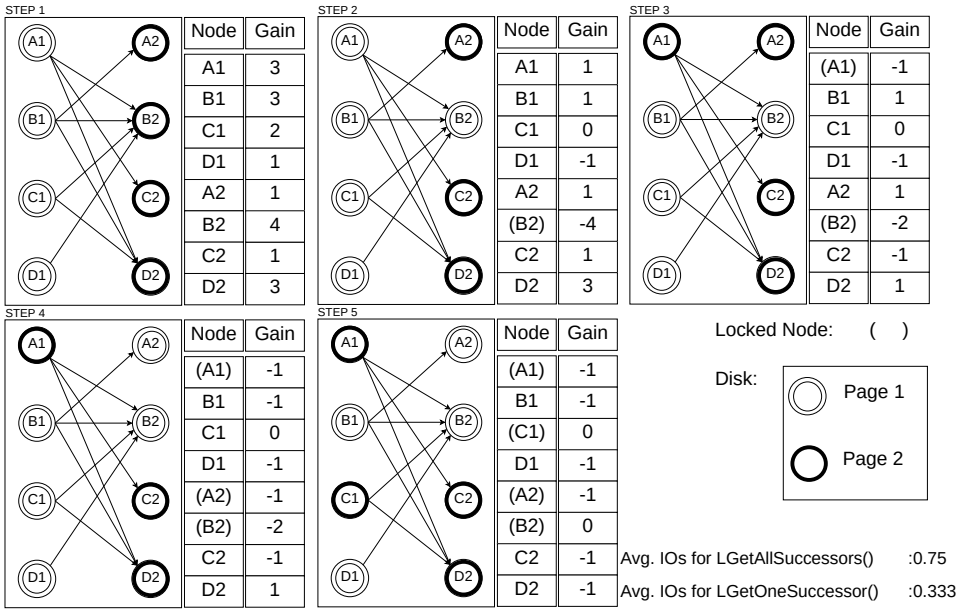


Figure 4.10: Two-way LCP-G1S

page. After moving $B2$, the partitions violate the node balancing constraint. Therefore, at Step 2, $A1$ is chosen to move to the other data page due to its having the largest gain. Again, $D2$ is chosen to move to the other partition at Step 3 due to its largest positive gain. We continue this process until no more positive gain takes place. Step 5 shows the partitioning result of one pass of the LCP-G∀S algorithm. As can be seen, the execution of one pass tries to minimize the SSTN-G∀S objective function from the initial solution. In contrast to LCP-G∀S, LCP-G1S relies on a min-cut gain function, as shown in Figure 4.10. We remind the reader that the two approaches have different objective (or gain) functions to optimize STN operations (e.g., $LGetAllSuccessors()$ and $LGetOneSuccessor()$). Even though the size of min-cuts by LCP-G∀S (Figure 4.9) is greater than for LCP-G1S (Figure 4.10), LCP-G∀S shows more reduced I/O costs for the $LGetAllSuccessors()$ operation due to the smaller $\sum_{n \in N} |LFS(n)|$.

Algorithm 4 shows a way to optimize our objective function with a K-way approach. The input is an initial partition with LCP-G1S (min-cut graph partitioning) and page size constraints. The output is disk storage of the STN network. Lines (1,2) show the stop criterion when no improvement is obtained over the best solution found. Line

Algorithm 4 Pseudocode for the K-way LCP-G∀S algorithm

Inputs:

- LCP-G1S-STN: partitioned STN with min-cuts
- PageSizeConstraints: maximum page size and minimum page utilization ratio
- G_{min}: minimum gain threshold value

Outputs:

- Data file containing STN data across data pages (LCP-G∀S-STN)

LCP-G∀S

```
1: totalGain = ∞
2: while totalGain > G_min do
3:    BN[] = LCP-G∀SBNs from STN
4:    for each node n in BN[] do
5:       curPNm = partition number of n
6:       NbrPNm[] = partition numbers of LCP-G∀SNbrs(n)
7:       check PageSizeConstraints and find maxGain from NbrPNm[]
8:       add maxGain into TB-LIST
9:    end for
10:   for each gain(n,curPNm,nbrPNm) in TB-LIST do
11:      try moving n from curPNm to nbrPNm
12:      lock node n
13:      update gains of LCP-G∀SNbrs(n) in TB-LIST
14:   end for
15:   find local maxima point and move all nodes until local maxima
16:   totalGain = local maxima gain
17: end while
```

(3) chooses possible candidates (LCP-G∀S boundary nodes) for solution modification. Lines (4-9) find the best movement for each candidate and create a Tabu-list. Lines (10-11) iteratively choose the best gain from the Tabu-list and temporarily move the node to explore a new solution. Line (12) locks the moving node to prevent revisiting the same solution in the pass. Line (13) updates the gains of $n \in LCP{-}G\forall SNbrs(n)$. Line (15) finds the local maxima and Lines (16,17) repeat the process until no further improvement is possible.

Table 4.2: SYMBOLS USED IN COST ANALYSIS

Symbol	Meaning
$\hat{E}$	Average edge/node ratio (node degree) in STN
$LRatio$	The probability that two nodes on a Lagrangian edge are stored into the same data page.
$\hat{SCE}$	Average distinct number of data pages across siblings

4.3.4 Analysis of LCP-G∀S algorithm

A Tabu search uses a Tabu-list in order to escape from the local maxima and search neighboring solutions until a certain stopping criterion is satisfied [79, 80]. Suppose that when Tabu search visits every solution, it assigns the current time-stamp to the solution. Then when a convergence Tabu Search (CTS) has visited all the neighboring solutions, it searches the earliest time-stamped neighbor [81, 82, 83]. The algorithm convergence of LCP-G∀S follows from a well-known result about the convergence of CTS [81].

Given a solution s, let $N(s)$ be the set of neighborhoods of s. Similarly, if $\acute{s} \in N(s)$, then $\acute{s}$ is a neighbor of s. Let $G_N = (V, A)$ be a graph induced by $N(s)$, where the node set V is the set of solutions E. The CTS algorithm converges and terminates after exploring all solutions E if the following two conditions hold [81]:

1. The neighborhood relation is symmetric, i.e. $x \in N(y) \Leftrightarrow y \in N(x)$ for all $x, y \in E$

2. Given a graph G_N, there exists a path between every pair of solution $x, y \in E$.

Lemma 12. *The LCP-G∀S algorithm converges and terminates.*

Proof. $STN_{p(n1)=A}$ and $STN_{p(n1)=B}$ shows the symmetric neighborhood relation. Moreover, every solution has a path to other solutions by moving a sequence of nodes. Since the LCP-G∀S satisfies the above two conditions, the proof is complete. □

Lemma 13. *The LCP-G∀S algorithm terminates after at most* $(N \cdot deg_{max})/G_{min}$ *passes.*

Proof. The LCP-G∀S uses G_{min} as a threshold. The termination condition (Step 2) is that the totalGain is below than G_{min} (threshold). Let N be the number of nodes. The sum of $LFS(n)$ for all nodes is at most $N \cdot (deg_{max} + 1)$ and at least N. In each pass, the totalGain should be greater than G_{min}. Therefore, the number of passes is bounded by $(N \cdot deg_{max})/G_{min}$. □

Lemma 14. *The time complexity of the LCP-G∀S algorithm is* $O(N \times deg_{max}^6 \times \#pass)$.

Proof. Let deg_{max} be the maximum node degree and N be the number of nodes. For simplicity, let us assume that the number of passes for the LCP-G∀S algorithm is $\#pass$. The runtime of the LCP-G∀S is dominated by two factors: gain computations and gain

updates. First, we only need to compute gains of LCP-G∀S boundary nodes (LCP-G∀SBNs). Since the number of these nodes is bounded by $O(N)$, we can choose at most N nodes as LCP-G∀SBNs. $LFS(n)$ takes $O(deg_{max})$ to compute its value. According to Proposition 2, the required time to compute a gain for one possible movement is $O(deg^2_{max})$ because the number of needed $LFS(n)$ is bounded by $O(deg_{max})$. According to Proposition 3, the number of $LCP{-}G\forall SNbrs(n)$ is bounded by $O(deg^2_{max})$. We only need to consider at most $O(deg^2_{max})$ target partitions to choose the largest gain among all possible movements. Therefore, $O(deg^4_{max})$ is required to compute the largest gain of a node. We assume that inserting, retrieving (or deleting), and updating a gain can be done in constant time since every gain is bounded above by $-deg_{max}$ and below by deg_{max} [78]. Thus, the total CPU time for gain computations is on the order of $O(N \times deg^4_{max})$. Second, whenever retrieving the highest gain in the Tabu-list, the node will be moved to another partition and locked. After the node n is moved, $LCP{-}G\forall SNbrs(n)$ should update their gains. The size of the Tabu-list is bounded by $O(N)$, and the number of $LCP{-}G\forall SNbrs(n)$ is bounded by $O(deg^2_{max})$. Since $O(deg^4_{max})$ is required to compute the largest gain of a node, the required time to update the gains of $LCP{-}G\forall SNbrs(n)$ is $O(deg^6_{max})$. Thus, the total CPU time for gain updates is on the order of $O(N \times deg^6_{max})$. We conclude that the total required time for the LCP-G∀S algorithm is $O(N \times deg^6_{max} \times \#pass)$. □

4.4 Analytical Evaluation And Cost Models

The goal of this section is to present cost models for estimating disk I/Os based on the STN operations in Table 4.1. Table 4.2 lists the symbols used to develop our cost formulas.

Traditional spatial networks use a connectivity ratio to measure predicated disk I/O [84]. We extended this connectivity ratio to formulate a spatio-temporal measurement we call the Lagrangian connectivity Ratio, or *LRatio*.

$$LRatio = \frac{Total\ number\ of\ unsplit\ Lagrangian\ edges}{Total\ number\ of\ Lagrangian\ edges} \tag{4.4}$$

The *LRatio* measures the connectivity along time and space in a STN. In Equation (4.4), Lagrangian edges refer to edges connecting nodes through time, such as the edges displayed in a TEG. This metric ignores the 'wait' edges in a TEG, so maximizing the

LRatio minimizes the disk I/Os for a *LGetOneSuccessor*() operation.

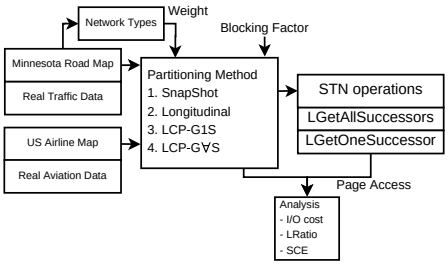
In this paper, we develop a new cost model for the *LGetAllSuccessors*() operation. To determine the expected number of page accesses for *LGetAllSuccessors*(), we need to compute the distinct number of data pages across siblings. We refer to this parameter as a Sibling Collocation Efficiency (SCE). $\hat{SCE}$ denotes the average SCE in a spatio-temporal network. $\hat{SCE}$ can be obtained by one scan of a set of successors on every node along the time series. Then we integrate $\hat{SCE}$ with the *LRatio*. Let us now consider the following case: Assume that the $FindNode()$ operation retrieves a source node and that a data page for the source node is located in the buffer cache. If *LGetAllSuccessors*() were then to retrieve all successors from the disk, the cost would be at most SCE I/Os. Then the probability that the data page of the source node is not contained in the data page set for its successors is $(1 - LRatio)^E$, resulting in a cost of one disk I/O for the $FindNode()$ operation. Therefore, the expected number of pages can now be obtained by integrating $\hat{SCE}$ and *LRatio* as shown in Equation (4.5).

$$Cost\ of\ LGetAllSuccessors() = \hat{SCE} + (1 - LRatio)^{\hat{E}} - 1 \tag{4.5}$$

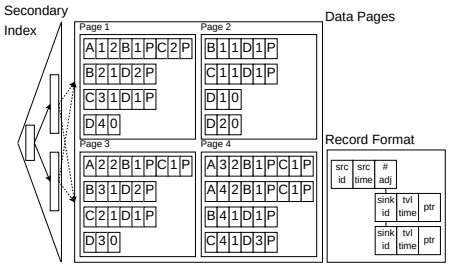
Table 4.3 summarizes cost models for STN operations. As can be seen, cost models for SSTN-G∀S and SSTN-G1S rely on the *LRatio*. However, the SSTN-G∀S cost model depends on another two parameters ($\hat{E}$ and $\hat{SCE}$). As expected, the two parameters are highly correlated to the edge/node ratio. For instance, if $\hat{E}$ becomes very large, then $(1 - LRatio)^E$ converges to 0. Therefore, $\hat{SCE}$ becomes the main factor that affects the performance of *LGetAllSuccessors*(). Consider the case where $\hat{E} = 1$; since $\hat{E}$ and $\hat{SCE}$ do not affect the performance for SSTN-G∀S (i.e., $\hat{E} = 1$ and $\hat{SCE} = 1$), the two cost models become exactly the same. Since SSTN-G1S is a good approximation to optimize SSTN-G∀S objective function, we choose LCP-G1S as an initial solution for LCP-G∀S and reoptimize the partitions to minimize SSTN-G∀S objective function.

Table 4.3: COST ANALYSIS FOR RETRIEVAL OPERATIONS

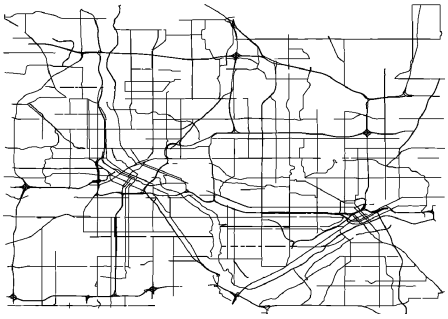
Operation	**Data page accesses**
$FindNode()$	1
$LGetAllSuccessors()$	$(1 - LRatio)^{\hat{E}} + \hat{SCE} - 1$
$LGetOneSuccessor()$	$1 - LRatio$



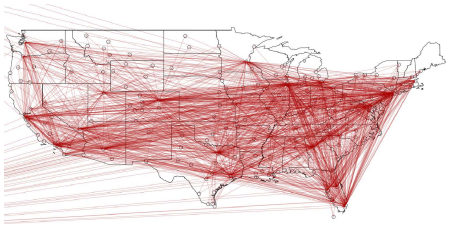
(a) Experimental Layout



(b) STN storage representation and record format



(c) Minneapolis Road Network



(d) US Major Airline Routes

Figure 4.11: Experimental Setup

4.5 Experimental Evaluation

The overall goal of the experiments is to show the performance improvements to access a set of successors that can be obtained by the LCP-G∀S algorithm. The metric of comparison in our experiments was the number of data pages accessed. We also define a blocking factor as the number of node records per data page. We wanted to answer six questions: (1) Can our cost models predict the disk I/Os for STN operations? (2) What is the effect of the blocking factor? (3) What is the effect of the number of time steps? (4) What is the effect of the edge/node ratio in STN? (5) What is the effect of higher edge/node ratio in real-world STN dataset? (6) What is the effect of node weights according to relative frequency of STN query?

4.5.1 Experimental Setup

Figure 4.11(a) shows our experimental setup. We used two real datasets: 1) a Minneapolis, MN road map consisting of $1,481$ nodes and $4,574$ edges (Figure 4.11(c)) and 2) a US major airline map with 226 nodes and $3,824$ edges (Figure 4.11(d)). MN roads are classified as either major or minor. Major roads have high traffic speeds and are often the main route to major destinations. We used real-world traffic datasets that consist of $5,760$ time steps and cover five days (Mon-Thu), obtained from the NAVTEQ corporation [85]. To understand the effect of parameter settings for LCP-G∀S, we also synthetically increased the edge/node ratio (node degree) by adding edges according to transitive closure of the edge relation on MN road map. Major US airline routes were chosen from the RITA dataset [86]. We used 240 time steps covering five days (01/01/12 to 01/05/12).

From these STN datasets, we created and stored four database files, using four different methods: LCP-G∀S, LCP-G1S, and two orthogonal partitioning methods (snapshot and longitudinal). These stored networks were then evaluated using two operations: $LGetAllSuccessors()$ and $LGetOneSuccessor()$. All experiments were performed on an Intel Core i7-2670QM CPU machine running MS Windows 7 with 8GB of RAM. Storage and access methods were implemented based on Java 1.7 and B+ secondary index distributed by [87].

4.5.2 Partitioning Methods

We performed experiments using four different STN storage candidates. The first two are orthogonal approaches (snapshot, longitudinal) and the third and fourth are LCP-G∀S and LCP-G1S. In our analysis, we used a stopping criterion for both the LCP-G∀S and LCP-G1S algorithms, described as follows: First, we defined a gain ratio (GR) as $\frac{Gain\ for\ one\ pass}{Cost\ of\ an\ initial\ partitioning}$. Then we ran several passes of the algorithms until the GR fell below a specified threshold value (GR_{min}). We set GR_{min} as $(0.1)^4$ for all experiments. In our implementation, both LCP-G∀S and LCP-G1S required around 30 passes to satisfy our stopping criterion. As an initial partitioning method, we used Metis [74]. Then LCP-G∀S and LCP-G1S were applied to the initial partitioning both to minimize their objective functions and preserve page-size constraints. Both LCP-G1S and LCP-G∀S algorithms were implemented based on Java 1.7.

4.5.3 STN Storage Representation and Record Format

Our STN storage model consists of two components: a secondary index and data pages. The secondary index (e.g., B+ tree or R tree) enables fast access to a data page using a data page pointer. A data page stores records using an adjacent record format. A record has a source-node id, a source-node time, and the number of incidents. Every incident has a node-id, a travel time, and a data page pointer for the incident record. For example, node $A1$ has two incidents (B, C) and occupies 9 record units in Figure 4.11(b). In our analysis, the default minimum page-space utilization ratio was 50% and the average page-space utilization ratio was 70%.

4.5.4 Experimental Observations and Results

Evaluation of Cost Model: The aim of the first set of experiments was to demonstrate the accuracy of our cost model proposed in Section 4.4. We used a uniform weight (i.e., all weights on nodes = 1) to simplify the interpretation of the results. The experiments used a blocking factor of 8 and chose randomly 50% of the total number of nodes. Table 4.4 summarizes the real and predicted disk I/Os of the STN operations with two real-world STN datasets. In our experiment, we used one buffer cache to store only one data page in memory and called a $FindNode()$ operation before calling a

Table 4.4: THE I/O COSTS OF STN OPERATIONS

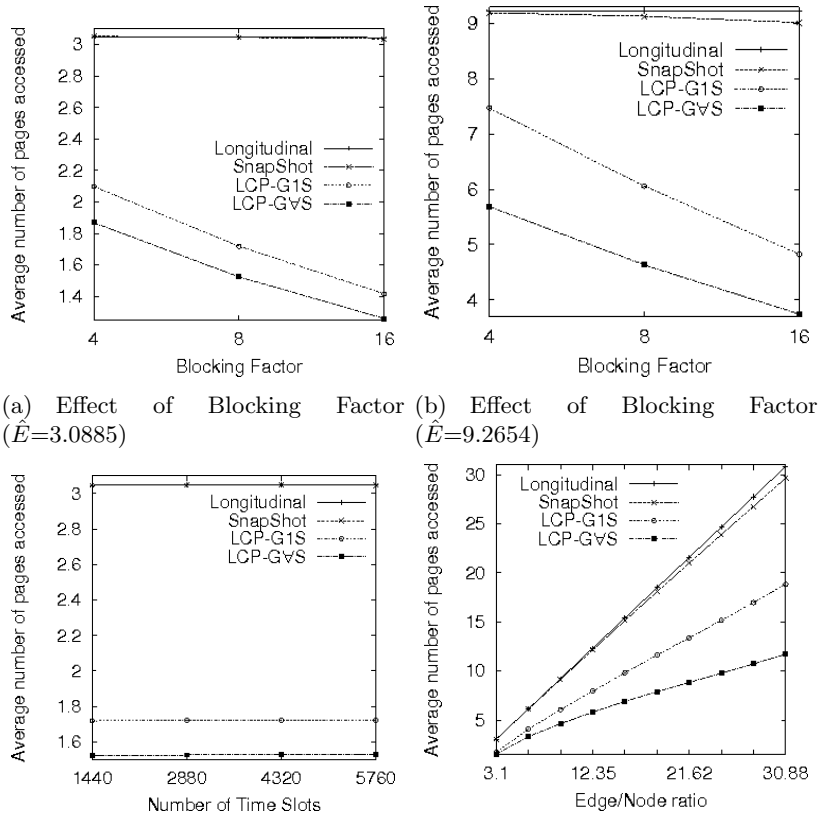
Operation / Method	LGetAllSuccessors()			LGetOneSuccessor()			*LRatio*	*SCE*
	Actual	Predicted	Pred. Error	Actual	Predicted	Pred. Error		
LCP-G∀S	1.5236	1.5150	0.57 %	0.6718	0.6718	0 %	0.3282	2.2222
LCP-G1S	1.7189	1.7192	0.02 %	0.5884	0.5886	0.04 %	0.4114	2.5245
SnapShot	3.0466	3.0470	0.01 %	1	1	0 %	0	3.0470
Longitudinal	3.0469	3.0474	0.01 %	1	1	0 %	0	3.0474
Road traffic dataset with time steps = 1,440, $\hat{E}$ = 3.0885, and blocking factor = 8								

Operation / Method	LGetAllSuccessors()			LGetOneSuccessor()			*LRatio*	*SCE*
	Actual	Predicted	Pred. Error	Actual	Predicted	Pred. Error		
LCP-G∀S	10.2470	10.2932	0.44 %	0.9788	0.9788	0 %	0.0212	10.6058
LCP-G1S	13.9845	14.0956	0.78%	0.9439	0.9438	0 %	0.0562	14.7320
SnapShot	16.4232	16.5919	1.01 %	1	1	0 %	0	16.5919
Longitudinal	16.8846	17.0747	1.11 %	1	1	0 %	0	17.0747
Aviation dataset with time steps = 240, $\hat{E}$ = 17.4898, and blocking factor = 8								

LGetAllSuccessors() or a *LGetOneSuccessor*(). As the table shows, the prediction error for our cost models is within 1%. When we called *LGetAllSuccessors*(), LCP-G∀S outperformed the other approaches. By contrast, when we called *LGetOneSuccessor*(), LCP-G1S performed best.

Effect of Blocking Factor: The second experiment evaluated the effect of the blocking factor on the performance of the algorithms. To evaluate the performance of alternative access methods, we worked with two STN datasets: one was a real Minnesota road map and the other, a synthetic road map with an increased edge/node ratio. We fixed a query set and increased the blocking factor. As shown in Figure 4.12(a) and 4.12(b), we observed an improvement in disk I/O efficiency for the two LCP methods over orthogonal methods, as expected, due to its ability to store temporally connected information on a single data page. A larger blocking factor enhanced disk I/O performance for both LCP-G∀S and LCP-G1S. This is because more data will be collocated based on Lagrangian-connectivity as the blocking factor increases. By contrast, we observed that longitudinal and snapshot showed no difference in disk I/Os.

Effect of Number of Time Steps: The third experiment evaluated the effect of the length of time series on the performance of the algorithms. We increased the number of time steps and therefore increased the number of data pages proportional to the increased number of records. The effect of the length of time steps on I/O costs is shown



(a) Effect of Blocking Factor ($\hat{E}$=3.0885)

(b) Effect of Blocking Factor ($\hat{E}$=9.2654)

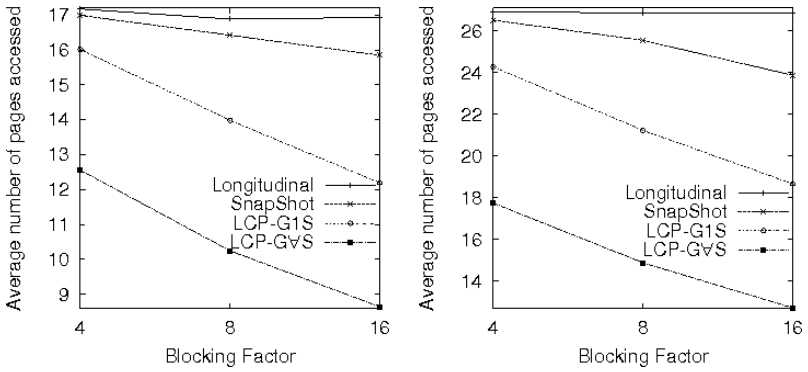
(c) Effect of Size of Time Slots ($\hat{E}$=3.0885)

(d) Effect of Edge/Node ratio (blocking factor=8)

Figure 4.12: Performance comparison for *LGetAllSuccessors*()

in Figure 4.12(c). As can be seen, time series length does not affect performance of any method. This property is desirable when a user incrementally stores large numbers of time steps because it allows us to carve a large number of time steps into tiny sections and store them individually.

Effect of Edge/Node ratio: The fourth experiment evaluated the effect of the edge/node ratio on network datasets. We synthetically connected nodes based on the transitive closure of a spatio-temporal directed graph and then increased the edge/node ratio. Figure 4.12(d) shows that a higher edge/node ratio increases the performance gap between LCP-G∀S and LCP-G1S, with LCP-G∀S performing better than LCP-G1S. These results demonstrate that the performance of LCP-G∀S relies on both the *LRatio* and $\hat{E}$, as shown in Table 4.3.



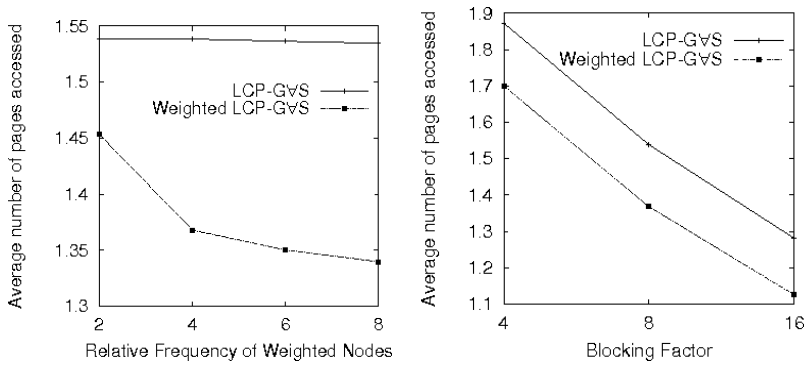
(a) 226 airports ($\hat{E} = 17.49$) (b) 120 airports ($\hat{E} = 27.80$)

Figure 4.13: Performance comparison with real aviation STN dataset

Aviation STN dataset: The fifth experiment evaluated the performance of the LCP-G∀S algorithm with a real-world aviation STN dataset. First, we chose 266 major airports, obtained from the RITA [86]. Then, we refined this selection to include more connected airports (120 airports) in terms of the number of flights between them. Figure 4.13(a) shows that LCP-G∀S reduced 27% disk I/Os for $LGetAllSuccessors()$ over LCP-G1S. When considering more connected STNs (Figure 4.13(b)), the performance gap between LCP-G∀S and LCP-G1S increases up to 30%.

Effect of Node Weight: The sixth experiment evaluated the effect of weights on the LCP-G∀S algorithm. First, we assigned a specific weight on major roads and used a unit weight (i.e., $w(n) = 1$) on minor roads. Then, query sets were chosen from the two types of roads according to an access frequency (weight). Figure 4.14(a) and 4.14(b) show that weighted LCP-G∀S decreases the I/O costs in the case of a given relative frequency. When considering $w(n) = 4$ for major roads and $w(n) = 1$ for minor roads, the Weighted LCP-G∀S reduced 9% disk I/Os for $LGetAllSuccessors()$ over non-weighted LCP-G∀S.

Construction time for LCP: In all the experiments, runtime of the LCP-G∀S algorithm was slower than LCP-G1S. This happens because LCP-G∀S had more neighbor nodes and needed to update more gains when moving a node. For example, when considering the real Minnesota road map with blocking factor 8, the number of partitions becomes $266,849$. In this case, our implementations showed that LCP-G∀S took 67



(a) Effect of Node Weight (blocking factor=8)

(b) Effect of Blocking Factor(Major Roads: $w(n) = 4$)

Figure 4.14: Performance comparison between LCP-G∀S and Weighted LCP-G∀S

minutes whereas LCP-G1S took 14 minutes. However, if we keep in mind that the objective function of LCP-G∀S differs from LCP-G1S, then we see that the reduced disk I/Os for *LGetAllSuccessors*() compensates for the overhead of construction time.

4.5.5 Summary of Results

In our experimental analysis, LCP-G∀S showed the best performance on the *LGetAllSuccessors*() operation against other approaches. Although the *LRatio* was a good approximation to optimize the operation, it showed a limitation to achieve the optimal partitioning. We defined the SSTN-G∀S objective function and showed a way to optimize the function according to our LCP-G∀S algorithm. In our I/O analysis, we introduced the notion of Sibling Collocation Efficiency (SCE) and formulated a cost model based on both *LRatio* and SCE. Because a higher edge/node ratio has more chance to minimize SCE, LCP-G∀S outperforms LCP-G1S when the edge/node ratio increases. When the edge/node ratio was 1, there was no difference between LCP-G∀S and LCP-G1S. When the edge/node ratio was 30.88, LCP-G∀S showed a 38% disk I/O reduction over LCP-G1S. In real-world road networks, the edge/node ratio was close to 3.0. Our results showed a 12% performance gain of a *LGetAllSuccessors*() operation over Minnesota road network (edge/node ratio = 3.0885). In the real-world airline network dataset, our results showed 27% performance gain over US major airline network (edge/node ratio = 17.4898). As a result, LCP-G∀S can minimize Lagrangian Family Set (LFS) to better

support *LGetAllSuccessors*() over STN datasets.

4.5.6 Discussion

LCP approaches, such as LCP-G∀S and LCP-G1S, can partition STN datasets according to STN connectivity and optimize STN operations. Recent work [66, 68, 69] on speed-up shortest path algorithms on STNs, such as hierarchical network partitioning, pre-computation, and bi-directional search, is being able to work orthogonally with the LCP approach. Even while our experimental analysis mainly relies on discrete STN data models (e.g., TEG), it can be extended to continuous STN data models (e.g., piece-wise linear cost model) if the travel time is non-uniform with a much more fine grained unit of time. In a recent paper [69], researchers use a continuous STN model with a snapshot approach that maintains the information of departure times that are closer to each other at the same disk data page. This speed-up shortest path algorithm may benefit from LCP approaches.

Our storage and access methods focus on general STN operations. We assume that STN datasets have high time resolution with non-uniform STN properties (e.g., turn restrictions, waiting time, road capacity, traffic congestion, and fuel consumption). STN datasets incorporating these STN proprieties may give more benefits to find intelligent routes (e.g., min-left turn routes, min-wait routes, and eco-routes) in future intelligent route planning systems.

4.6 Conclusion and future work

Spatio-temporal networks are becoming increasingly important for a variety of societal applications such as transportation management, fuel distribution, airline routing, and electrical grid usage analysis. Traditional orthogonal-based storage approaches of STN data produce significant I/O costs when performing spatio-temporal network queries. We propose LCP-G∀S and LCP-G1S methods to efficiently store and access spatio-temporal networks that use the spatio-temporal interaction between nodes and edges in a network. Experimental evaluation of our approaches demonstrated significant improvements over orthogonal approaches.

There are several interesting directions for future work. First, we would like to

investigate novel indexing techniques for STNs. Second, we intend to extend the Lagrangian reference framework to model hierarchical networks. In particular, hierarchical STN structures may utilize Lagrangian partitions to reduce the computational time for STN routing algorithms. Lastly, for simplicity, LCP-G∀S and LCP-G1S are iterative algorithms. In the future, developing parallel algorithms of these methods may be more efficient to reduce the computation time.

Chapter 5

Conclusion and Future Work

Spatial Network Big Data is important in several societal application domains such as transportation safety, public safety, disaster response and recovery, and surface and air transportation management systems. The most important requirement of SNBD is to timely utilize spatial and spatio-temporal network datasets, including GPS trace data from cell-phones, mobile datasets, VGI, and crowd-sourcing. However, SNBD poses several significant challenges for current spatial network computing techniques and spatial network database systems. In this thesis, I investigated three SNBD problems and address these challenges.

5.1 Key Results

This section presents a summary of the major results that were produced as a part of this thesis.

5.1.1 Capacity Constrained Network Voronoi Diagram

We presented the problem of creating a Capacity Constrained Network Voronoi Diagram (CCNVD). An important potential application of CCNVD is promoting transportation resiliency after a disaster. Creating a CCNVD is challenging because of the large size of the transportation network and the constraint that service areas must be contiguous in the graph to simplify communication of service center allotments. In this work, we describe our Pressure Equalizer (PE) approach for creating a CCNVD that meets the

capacity constraints of service centers while maintaining the contiguity of service areas assigned to those centers. Improving PE's scalability to large sized transportation networks requires addressing the computational bottleneck that occurs during Service Area Contiguity Checking (SACC). To remedy the problem, we proposed a novel SACC algorithm, namely Block Tree Contiguity Checking (BTCC), to reduce the computational cost of the PE algorithm. Experiments using five different transportation networks demonstrated that our proposed algorithms significantly reduce the computational cost for CCNVD.

5.1.2 Evacuation Route Planning

Evacuation route planning for large transportation networks is becoming increasingly important for dealing with man-made and natural disasters, such as hurricanes, terrorist acts, and nuclear accidents. An important component of evacuation planning methods is the ability to account for capacity constraints of the road network with manageable computational cost. In this work, We introduced a dartboard network structure to reflect an evacuee flow pattern for common evacuation scenarios by exploiting the spatial structure of the road network. Our DBNC-ERP algorithm partitions the network using dartboard network cuts (DBN-cuts) and groups source nodes in different spatial locations to maximize the number of evacuees. We also showed the cost model to explain how to reduce the computational cost based on dartboard network structure. Experimental evaluation of DBNC-ERP demonstrated significant improvements over previous work.

5.1.3 Storing Spatio-Temporal Network

Spatio-temporal networks are becoming increasingly important for a variety of societal applications such as transportation management, fuel distribution, airline routing, and electrical grid usage analysis. Traditional orthogonal-based storage approaches of STN data produce significant I/O costs when performing spatio-temporal network queries. We propose LCP-G$\forall$S and LCP-G1S methods to efficiently store and access spatio-temporal networks that use the spatio-temporal interaction between nodes and edges

in a network. Experimental evaluation of the approaches demonstrated significant improvements over orthogonal approaches.

5.2 Future Directions

We plan to extend the current research and make efforts to design novel Spatial network Big Data (SNBD) database systems.

5.2.1 Conceptual Level

Entity-Relationship Diagrams (ERD) and pictograms are widely used in conceptual modeling to represent spatial network data types, their relationships and the associated constraints. Figure 5.1 shows an example of a pictogram that represents a spatio-temporal network. However, these models have a limitation to fully express the spatial and spatio-temporal network structures (e.g., spatio-temporal topological relationship, spatio-temporal hotspots, etc). We plan to investigate novel SBD conceptual data models to handle these limitations.

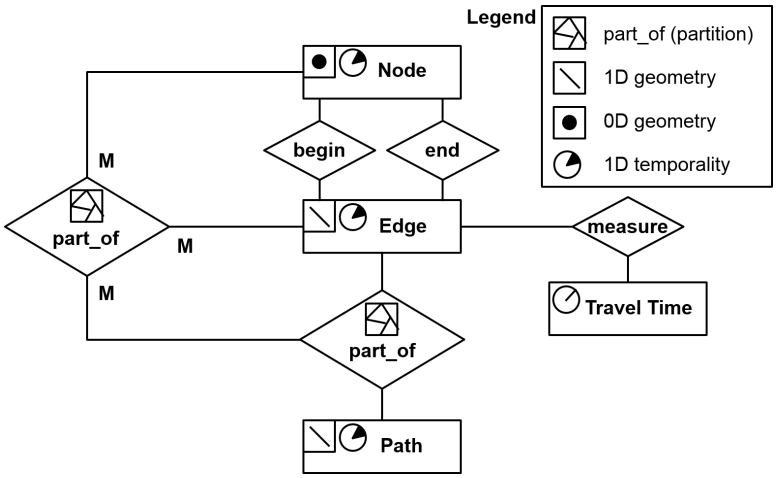


Figure 5.1: Pictogram of a spatio-temporal network

5.2.2 Logical Level

SNBD Logical Data Model

Previous work on SNBD logical data models can be categorized into two groups: (1) Time-Expanded Graph (TEG) and (2) Time-Aggregated Graph (TAG). TEG replicates each node along the time series such that a time-varying attribute is represented between replicated nodes. In a TEG, every node is associated with a temporal attribute, and every edge joining two nodes represents the spatio-temporal relation between the two nodes. However, the size of the TEG increases proportionally to the number of the time-steps. A TAG models the changes in STNs by collecting the node and edge attributes into a set of time series. Even though TAGs can reduce the storage space for large-sized STN datasets, they have limitations for representing big-sized STN datasets.

We plan to investigate novel logical data models to efficiently represent STNs. One feasible goal is to develop a Lagrangian-Aggregated Graph (LAG) which groups the node and edge attributes through Lagrangian connectivity. In addition, we would like to develop Sub-Network-Aggregated Graph (SNAG) which would identify similar spatio-temporal sub-networks in SNBDs and group these sub-networks to simplify the representation of STNs.

Table Name	Description
Node	create table node_table (node char, facility_name varchar(100), time-step time)
Edge	create table edge_table (node_a char, node_b char, distance int, capacity int, time-step time)

(a) Tables for nodes and edges

	Function	Example
Shortest Path	STN_dijkstra (text sql, char source, char target, time start-time)	STN_dijkstra ('select node_a, node_b, distance from edge_table', 'A', 'Z', 1)
Max Flow	STN_maxflow (text sql, char source, char target, text time-interval)	STN_maxflow ('select node_a , node_b, capacity from edge_table', 'A', 'Z', '1-9')
Min Cost Flow	STN_mcf (text sql, char source, char target, time start-time)	STN_mcf ('select node_a, node_b, capacity, distance from edge_table', 'A', 'Z', 1)
Network Voronoi Diagram	STN_nvd (text sql1, text sql2, time start-time)	STN_nvd ('select node a, node b, distance from edge_table', 'select node from node _table where facility_name='shelter'', 1)
Min-cut	STN_min_cut (text sql, text time-interval)	STN_min_cut ('select node_a, node_b from edge_table', '1-9')

(b) Spatio-Temporal Network Queries

Figure 5.2: Example of an SNBD Query Languages

SNBD Query Language

Existing query languages, such as SQL, OQL, SQL3, etc, have not been designed to query spatio-temporal network datasets. For example, although SQL uses a 'start with .. connect by' clause to traverse graph datasets, it requires additional tables and complex inline queries to compute the spatio-temporal shortest path between two nodes. Figure 5.2 shows possible proposed examples of Spatio-temporal Network Queries. We plan to investigate graph algorithms and design SNBD query language for spatio-temporal network analysis.

5.2.3 Physical Level

SNBD Query Processing Strategy

Crowdsourced data is a growing component of SNBD. We would like to explore crowdsourced datasets and develop novel analytic and computational algorithms discovering non-trivial patterns (e.g., spatio-temporal network hotspots). In addition, we are also interested in transportation planning, particularly how cities can ensure safe and efficient transportation networks that are also cost-efficient and resilient in the wake of a disaster. Computational methods to address such issues could potentially have a wide range of societal applications in the future.

SNBD Storage Model

First, we would like to investigate novel indexing techniques for SNBD. Second, we intend to extend the Lagrangian reference framework to model hierarchical networks. In particular, hierarchical STN structures may utilize Lagrangian-path partitions to reduce the computational time for SNBD routing algorithms. Lastly, we would like to continue the development of more scalable and reliable database platforms for SNBD. Recently, cloud database platforms such as MapReduce [88], Apache Hadoop [89], Apache Spark [90] and GraphLab [91] have emerged as important database technologies for large-scale data-parallel applications such as data mining and scientific computing. The main bottleneck of these platforms for SNBD is disk I/Os due to data being shared across wide geographical areas and among different machines. We are interested in developing new data partitioning methods for SNBD that could reduce the bottleneck in

cloud database platforms and balance the workload among distributed machines.

References

[1] Shashi Shekhar, Viswanath Gunturi, Michael R Evans, and KwangSoo Yang. Spatial big-data challenges intersecting mobility and cloud computing. In *Proceedings of the Eleventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*, pages 1–6. ACM, 2012.

[2] James Manyika, Michael Chui, Brad Brown, Jacques Bughin, Richard Dobbs, Charles Roxburgh, Angela Hung Byers, and McKinsey Global Institute. Big data: The next frontier for innovation, competition, and productivity. 2011.

[3] Waze. `https://www.waze.com/`.

[4] Uber. `https://www.uber.com/`.

[5] ABC News(2012). Hurricane sandy's aftermath: Long lines at gas stations. `http://goo.gl/omQ62`,Retrieved Mar. 2013.

[6] S. Turner, R. Margiotta, and T. Lomax. Lessons learned: monitoring highway congestion and reliability using archived traffic detector data. *US Department of Transportation, Federal Highway Administration*, 2004.

[7] B. George and S. Shekhar. Time-aggregated graphs for modeling spatio-temporal networks. *Journal on Data Semantics XI*, pages 191–212, 2008.

[8] K Yang, Michael R Evans, Venkata MV Gunturi, James M Kang, and Shashi Shekhar. Lagrangian approaches to storage of spatio-temporal network datasets. *Knowledge and Data Engineering, IEEE Transactions on*, 26(9):2222–2236, 2014.

[9] Ramez Elmasri and Shamkant B Navathe. *Fundamentals of database systems*. Pearson, 2014.

[10] Q. Lu, B. George, and S. Shekhar. Capacity constrained routing algorithms for evacuation planning: A summary of results, Proceedings of 9th International Symposium on Spatial and Temporal Databases, LNCS 3633. pages 291–307. Springer, 2005.

[11] KwangSoo Yang, Venkata MV Gunturi, and Shashi Shekhar. A dartboard network cut based approach to evacuation route planning: A summary of results. In *Geographic Information Science*, pages 325–339. Springer, 2012.

[12] Shashi Shekhar, KwangSoo Yang, Venkata MV Gunturi, Lydia Manikonda, Dev Oliver, Xun Zhou, Betsy George, Sangho Kim, Jeffrey MR Wolff, and Qingsong Lu. Experiences with evacuation route planning algorithms. *International Journal of Geographical Information Science*, 26(12):2253–2265, 2012.

[13] KwangSoo Yang et al. Capacity-constrained network-voronoi diagram: A summary of results. In *Proceedings of the 13th International Conference on Advances in Spatial and Temporal Databases*, SSTD'13, pages 56–73, Berlin, Heidelberg, 2013. Springer-Verlag.

[14] Scientific American. New obama climate change order aims to prepare communities for severe weather. `http://goo.gl/W4x7oa`, Retrieved Nov 1, 2013.

[15] Medline. Gibbs LI and others. Hurricane sandy after action: Report and recommendations to mayor michael r. bloomberg. `goo.gl/v9wC66`, Retrieved May 2013.

[16] M.S. Daskin. *Network and discrete location: models, algorithms, and applications*. Wiley-Interscience series in discrete mathematics and optimization. Wiley, 2013.

[17] ZF Reza and H Masoud. *Facility Location: Concepts, Models, Algorithms and Case Studies*. Springer Dordrecht Heidelberg London New York, 2009.

[18] M.E. Dyer et al. On the complexity of partitioning graphs into connected subgraphs. *Discrete Applied Mathematics*, 10(2):139 – 153, 1985.

[19] Ajit A. Diwan. Partitioning into connected parts, (slide 7) in "graph partitioning problems". Research Promotion Workshop on Introduction to Graph and Geometric Algorithms, January 2011. `http://goo.gl/b8fTN`.

[20] Dominique Barth et al. A degree bound on decomposable trees. *Discrete mathematics*, 306(5):469–477, 2006.

[21] Reinhard Diestel. *Graph Theory, 4th Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2005.

[22] Robert Tarjan. Depth-first search and linear graph algorithms. In *Switching and Automata Theory, 1971., 12th Annual Symposium on*, pages 114–121, 1971.

[23] Michael T Gastner et al. The spatial structure of networks. *The European Physical Journal B-Condensed Matter and Complex Systems*, 49(2):247–252, 2006.

[24] Dov Harel et al. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[25] Michael A. Bender et al. The lca problem revisited. In *Proceedings of the 4th Latin American Symposium on Theoretical Informatics*, LATIN '00, pages 88–94, London, UK, UK, 2000. Springer-Verlag.

[26] Stephen Alstrup et al. Nearest common ancestors: a survey and a new distributed algorithm. In *SPAA*, pages 258–264, 2002.

[27] Johannes Fischer et al. Theoretical and practical improvements on the rmq-problem, with applications to lca and lce. In *Proceedings of the 17th Annual conference on Combinatorial Pattern Matching*, CPM'06, pages 36–48, Berlin, Heidelberg, 2006. Springer-Verlag.

[28] Andrej Brodnik et al. Membership in constant time and almost-minimum space. *SIAM Journal on Computing*, 28(5):1627–1640, 1999.

[29] Dimitris Fotakis et al. Space efficient hash tables with worst case constant access time. *Theory of Computing Systems*, 38(2):229–248, 2005.

[30] Jeffery Westbrook et al. Maintaining bridge-connected and biconnected components on-line. *Algorithmica*, 7(1-6):433–464, 1992.

[31] Zvi Galil and GiuseppeF. Italiano. Maintaining biconnected components of dynamic planar graphs. In JavierLeach Albert et al., editors, *Automata, Languages and Programming*, volume 510 of *Lecture Notes in Computer Science*, pages 339–350. Springer Berlin Heidelberg, 1991.

[32] Camil Demetrescu et al. In Mikhail J. Atallah et al., editors, *Algorithms and Theory of Computation Handbook*, chapter Dynamic Graph Algorithms, pages 9–9. Chapman & Hall/CRC, 2010.

[33] Michael L Fredman et al. Fibonacci heaps and their uses in improved network optimization algorithms. *Journal of the ACM (JACM)*, 34(3):596–615, 1987.

[34] B.B.H. Korte et al. *Combinatorial Optimization*. Algorithms and Combinatorics. Springer-Verlag Berlin Heidelberg, 2012.

[35] OpenStreetMap. `http://goo.gl/Hso0`,Retrieved Jul. 2013.

[36] George Karypis et al. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on scientific Computing*, 20(1):359–392, 1998.

[37] David S Johnson et al. *Network flows and matching: first DIMACS implementation challenge*, volume 12. Amer Mathematical Society, 1993.

[38] Atsuyuki Okabe et al. *Spatial Analysis Along Networks: Statistical and Computational Methods*. Statistics in Practice. Wiley, 2012.

[39] Martin Erwig. The graph voronoi diagram with applications. *Networks*, 36(3):156–163, 2000.

[40] Atsuyuki Okabe et al. Generalized network voronoi diagrams: Concepts, computational methods, and applications. *International Journal of Geographical Information Science*, 22(9):965–994, 2008.

[41] R.K. Ahuja et al. *Network flows: theory, algorithms, and applications*. Prentice Hall, 1993.

[42] The New York Times, HURRICANE ANDREW: When a Monster Is on the Way, 'It's Time to Get Out of Town'; In Texas, a Line of Cars 50 Miles Long, August 26, 1992, `http://goo.gl/hq0EH`, Retrieved Apr. 2012.

[43] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, and K. Weihe. Network flows: theory, algorithms and applications. Prentice Hall. 1993.

[44] A. Schrijver. *Combinatorial optimization*. Springer, 2003.

[45] F.S. Hillier, G.J. Lieberman, and M. Hillier. *Introduction to operations research*. McGraw-Hill, 1990.

[46] H.W. Hamacher and S.A. Tjandra. Mathematical modelling of evacuation problems: State of the art. Pedestrian and Evacuation Dynamics. pages 227–266. Springer, 2002.

[47] M. Ben-Akiva et al. Development of a deployable real-time dynamic traffic assignment system: Dynamit and dynamit-p users guide. *Intelligent Transportation Systems Program, Massachusetts Institute of Technology*, 2002.

[48] H.S. Mahmassani, H. Sbayti, and X. Zhou. Dynasmart-p: Intelligent transportation network planning tool: Version 1.0 users guide. *College Park, MD: Maryland Transportation Initiative, University of Maryland*, 2004.

[49] DR Ford and D.R. Fulkerson. *Flows in networks*. Princeton university press, 2010.

[50] L. Fleischer and M. Skutella. Quickest flows over time. SIAM Journal on Computing. volume 36, pages 1600–1630. [Philadelphia] Society for Industrial and Applied Mathematics., 2007.

[51] J.G. Wardrop. Some theoretical aspects of road traffic research. In *Proceedings of the Institution of Civil Engineers, 2(1)*. Thomas Telford Ltd., 1952.

[52] X. Zhou, B. George, S. Kim, J.M.R. Wolff, Q. Lu, S. Shekhar, O. Nashua, and G. Team. Evacuation planning: A spatial network database approach. bulletin of the technical committee on data engineering, vol.33 no.2. page 26. IEEE Computer Society, 2010.

[53] JW Suurballe. Disjoint paths in a network. Networks. volume 4, pages 125–145. Wiley, 1974.

[54] D. Sidhu, R. Nair, and S. Abdallah. Finding disjoint paths in networks. In *ACM SIGCOMM Computer Communication Review*, volume 21, pages 43–51. ACM, 1991.

[55] Jon Michael Kleinberg. Approximation algorithms for disjoint paths problems. *Ph.D. Dissertation, Dept. of CS., Massachusetts Institute of Technology*, 1996.

[56] R.E. Korf, W. Zhang, I. Thayer, and H. Hohwald. Frontier search. Journal of the ACM (JACM). volume 52, pages 715–748. ACM, 2005.

[57] F. Xie and D. Levinson. Measuring the structure of road networks. Geographical Analysis. volume 39, pages 336–356. Wiley, 2007.

[58] S. Shekhar and S. Chawla. Spatial databases: a tour. *Upper Saddle River, New Jersey, Prentice Hall*, 7458, 2003.

[59] D. Levinson and B. Yerra. Self-organization of surface transportation networks. Transportation Science. volume 40, pages 179–188. INFORMS, 2006.

[60] J.W. Suurballe and R.E. Tarjan. A quick method for finding shortest pairs of disjoint paths. Networks. volume 14, pages 325–336. Wiley, 1984.

[61] Ramesh Bhandari. *Survivable Networks: Algorithms for Diverse Routing*. Kluwer Academic Publishers, Norwell, MA, USA, 1998.

[62] U.S.Census Bureau - TIGER/Lines, `http://goo.gl/P6Ye7`, Retrieved Jan. 2012.

[63] Michael R. Evans, KwangSoo Yang, James M. Kang, and Shashi Shekhar. A lagrangian approach for storage of spatio-temporal network datasets: a summary of results. In *GIS*, pages 212–221, 2010.

[64] Betsy George, Sangho Kim, and Shashi Shekhar. Spatio-temporal network databases and routing algorithms: A summary of results. In *SSTD*, pages 460–477, 2007.

[65] D. Delling, P. Sanders, D. Schultes, and D. Wagner. Engineering route planning algorithms. *Algorithmics of large and complex networks*, pages 117–139, 2009.

[66] Ugur Demiryurek, Farnoush Banaei Kashani, and Cyrus Shahabi. Efficient k-nearest neighbor search in time-dependent spatial networks. In *DEXA (1)*, pages 432–449, 2010.

[67] Venkata M. V. Gunturi, Ernesto Nunes, KwangSoo Yang, and Shashi Shekhar. A critical-time-point approach to all-start-time lagrangian shortest paths: A summary of results. In *SSTD*, pages 74–91, 2011.

[68] Ugur Demiryurek, Farnoush Banaei Kashani, Cyrus Shahabi, and Anand Ranganathan. Online computation of fastest path in time-dependent spatial networks. In *SSTD*, pages 92–111, 2011.

[69] Lívia A Cruz, Mario A Nascimento, and José AF de Macêdo. k-nearest neighbors queries in time-dependent road networks. *Journal of Information and Data Management*, 3(3):211, 2012.

[70] M.F. Mokbel, T.M. Ghanem, and W.G. Aref. Spatio-temporal access methods. *IEEE Data Engineering Bulletin*, 26(2):40–49, 2003.

[71] G.K. Batchelor. *An introduction to fluid dynamics*. Cambridge Univ Pr, 2000.

[72] L. R Ford and D. R Fulkerson. Constructing maximal dynamic flows from static flows. In *OPERATIONS RESEARCH Vol. 6, No. 3, May-June 1958, pp. 419-433 DOI: 10.1287/opre.6.3.419*, 1958.

[73] S. Pallottino and M.G. Scutella. Shortest path algorithms in transportation models: classical and innovative aspects. *Equilibrium and advanced transportation modelling*, pages 245–281, 1998.

[74] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 20(1):359, 1999.

[75] C.J. Alpert and A.B. Kahng. Recent directions in netlist partitioning: a survey* 1. *INTEGRATION, the VLSI journal*, 19(1-2):1–81, 1995.

[76] Michael R. Garey and David S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990.

[77] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *Bell System Technical Journal*, 49(2):291–307, 1970.

[78] C.M. Fiduccia and R.M. Mattheyses. A linear-time heuristic for improving network partitions. In *Design Automation, 1982. 19th Conference on*, pages 175–181. IEEE, 1982.

[79] F. Glover. Tabu search-part ii. *ORSA Journal on computing*, 2(1):4–32, 1990.

[80] W. Michiels, E.H.L. Aarts, and J. Korst. *Theoretical aspects of local search*. Springer-Verlag New York Inc, 2007.

[81] Said Hanafi. On the convergence of tabu search. *Journal of Heuristics*, 7(1):47–58, 2001.

[82] Fred Glover and Said Hanafi. Tabu search and finite convergence. *Discrete Appl. Math.*, 119(1-2):3–36, June 2002.

[83] F. Glover and S. Hanafi. *Progress in Combinatorial Optimization*, volume 486, chapter 14. CONVERGENT TABU SEARCH FOR OPTIMAL PARTITIONING, pages 423–442. Wiley-ISTE, February 2010.

[84] S. Shekhar and D.R. Liu. CCAM: A connectivity-clustered access method for networks and network computations. *IEEE Transactions on Knowledge and Data Engineering*, 9(1):102–119, 1997.

[85] NAVTEQ, `www.navteq.com`.

[86] RITA, `www.transtats.bts.gov`.

[87] JDBM, Open-source B+Tree, jdbm.sourceforge.net.

[88] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2008.

[89] Konstantin Shvachko, Hairong Kuang, Sanjay Radia, and Robert Chansler. The hadoop distributed file system. In *Mass Storage Systems and Technologies (MSST), 2010 IEEE 26th Symposium on*, pages 1–10. IEEE, 2010.

[90] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: cluster computing with working sets. In *Proceedings of the 2nd USENIX conference on Hot topics in cloud computing*, pages 10–10, 2010.

[91] Yucheng Low, Danny Bickson, Joseph Gonzalez, Carlos Guestrin, Aapo Kyrola, and Joseph M Hellerstein. Distributed graphlab: a framework for machine learning and data mining in the cloud. *Proceedings of the VLDB Endowment*, 5(8):716–727, 2012.